MMA955xL Intelligent, Motion-Sensing Platform Software Reference Manual

Devices Supported:

MMA9550L

MMA9551L

MMA9553L

Document Number: MMA955XLSWRM
Rev. 0, 04/2012

MMA955XLSWRM
Rev. 0
04/2012

Contents

Chapter 1 About This Document

1.1 Overview

1.1.1 Purpose

This software reference manual describes the features, architecture, and programming model of the MMA9550L, MMA9551L, and MMA9553L intelligent, motion-sensing platforms. Additional information for the MMA9553L platform is available in the *MMA9553L Intelligent Pedometer Platform Software Reference Manual* (MMA9553LSWRM). (See item 1 in "References" on page 13.)

The MMA9559L platform is a member of the MMA955xL family that has the same hardware as the other platforms, but significantly different firmware. For that reason, the MMA9559L platform is described in a separate software reference manual (MMA9559LSWRM). (See item 1 in "References" on page 13.)

1.1.2 Audience

This document is primarily for system architects and software application developers who are using or considering use of the MMA9550L, MMA9551L, and MMA9553L platforms in a system.

1.2 Terms and acronyms

AFE	Analog Front End
APP_ID	Application Identifier
API	Application Programming Interface
CC	Command Complete
CI	Command Interpreter
CMD	Command
COCO	Conversion Complete
DFC	Data Format Code
DTAP	Double tap (n.)
FIFO	First In First Out, a data structure
GPIO	General-Purpose Input/Output, a microcontroller pin that can be programmed by software
HG	High g
I^2C	Inter-integrated circuit
LG	Low g

LL	Landscape left
LR	Landscape right
MAC	Multiply-accumulate unit
MBOX	Mailbox
MCU	Microcontroller
MEMS	Microelectromechanical systems
MISO	SPI Master In, Slave Out
MOSI	SPI Master Out, Slave In
MTIMOV	Module Timer Overflow Module
PD	Portrait Down
PDB	Program Delay Block
PL	Portrait/Landscape
Platform	A grouping of hardware and software, working together
PU	Portrait Up
SFD	Start Frame Digital
SPI	Serial, peripheral interface
TPM	Timer Program Module

1.3 Conventions

This document uses the following notational conventions:

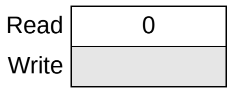
cleared/set	When a bit takes the value 0, it is said to be cleared; when it takes a value of 1, it is said to be set.
MNEMONICS	In text, instruction mnemonics are shown in uppercase.
`mnemonics`	In code and tables, instruction mnemonics are shown in lowercase.
italics	Italics indicate variable command parameters. Book titles also are italicized.
0x*0*	The *0x* prefix to denote a hexadecimal number
*0*b	The *b* suffix to denote a binary number
REG[FIELD]	Abbreviations for registers are shown in uppercase. Specific bits, fields or ranges appear in brackets. For example, RAMBAR[BA] identifies the base address field in the RAM base-address register.
nibble	A four-bit data unit
byte	An eight-bit data unit
word	A 16-bit data unit
longword	A 32-bit data unit
x	In some contexts, such as signal encodings, x indicates a “do not care.”
n	Used to express an undefined numerical value.
~	NOT logical operator
&	AND logical operator
|	OR logical operator
||	Field concatenation operator
$\overline{\text{OVERBAR}}$	Indicates that a signal is active-low.

1.4 Register figure conventions

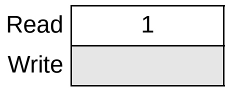
This document uses the following conventions for the register reset values:

—	The bit is undefined at reset.
u	The bit is unaffected by reset.
[*signal_name*]	Reset value is determined by the polarity of the indicated signal.

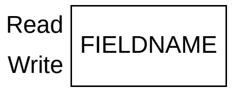
The following register fields are used:



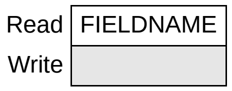
Indicates a reserved bit field in a memory-mapped register. These bits are always read as 0.



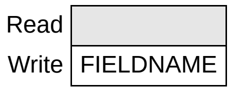
Indicates a reserved bit field in a memory-mapped register. These bits are always read as 1.



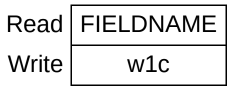
Indicates a read/write bit.



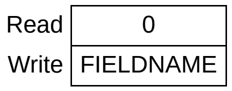
Indicates a read-only bit field in a memory-mapped register.



Indicates a write-only bit field in a memory-mapped register.



Write 1 to clear: indicates that writing a 1 to this bit field clears it.



Indicates a self-clearing bit.

1.5 References

1. MMA955xL Intelligent Motion-Sensing Platform documentation: “MMA955xL: Product Documentation Page”
2. IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std. 1149.1™-2001 (R2008)
3. The I^{2}C-Bus Specification Version 2.1, January 2000, Philips Semiconductors
4. I^{2}C-Bus Specification and User Manual, NXP Semiconductors Document UM10204, Rev. 03 - 19 June 2007
5. ColdFire Family Programmer’s Reference Manual, Freescale Semiconductor, CFPRM Rev. 3, 03/2005
6. Wikipedia entry for “Semaphore”: http://en.wikipedia.org/wiki/Semaphore_(programming)
7. *ITU-T V.41 Recommendation: Code-Independent Error Control System*, available at http://www.itu.int/publications/index.html.
8. *ITU-T X.25 Recommendation: Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit*, available at http://www.itu.int/publications/index.html.
9. *ITU-T T.30 Recommendation: Procedures for document facsimile transmission in the general switched telephone network*, available at http://www.itu.int/publications/index.html.

Chapter 2 Introduction

The MMA955xL low-g, 3-axis accelerometer is a member of Freescale's Xtrinsic family of intelligent sensor platforms.

The device incorporates dedicated accelerometer MEMS transducers, signal-conditioning, data conversion, a 32-bit programmable microcontroller, and flexible communications and I/O pins.

This unique blend of capabilities transforms the MMA955xL platform into an intelligent, high-precision, motion-sensing platform able to manage multiple sensor inputs and make the system-level decisions required for sophisticated applications such as gesture recognition and pedometer tasks.

The MMA955xL platform can be further programmed and configured with the CodeWarrior Development Studio software. C, C++, and ColdFire assembly programming languages are supported. (For more information, see "References" on page 13.) This Eclipse-based, integrated-design environment enables users to quickly and easily shape and implement custom algorithms and features to exactly match their project needs.

The MMA955xL platform can be used in conjunction with a host processor in any system that requires data acquisition and processing in response to motion of the entire system. The host processor runs the host application and the MMA955xL platform runs the embedded application that provides a data interface to acceleration data from the linear acceleration sensor.

2.1 Functional overview

A host processor communicates with the MMA955xL devices via I^2C or SPI serial buses. A program on the host sends the commands via either the I2C or SPI serial bus and the MMA955xL platform's Command Interpreter (CI) interprets and responds to those commands.

The platform has two CIs—one ROM-based and one flash-based. The most-recent reset determines which memory space the device is in. If the device is in ROM memory space, it runs the ROM CI. If the device is in flash memory and runs the flash CI.

The ROM-based CI commands are simpler, low-level commands. The flash-based CI commands are higher-level, higher-function commands.

The ROM-CI commands are described in the MMA955xL *Intelligent, Motion-Sensing Platform Hardware Reference Manual* (MMA955xLRM). The FLASH-CI commands are described in this document.

The MMA955xL device can also be configured and programmed to act as a bus master. The device talks to secondary sensors—such as pressure sensors, magnetometers, or gyroscopes—through its master serial communication interface. The MMA955xL platform's master serial port can operate in I^2C or SPI mode. By offloading the secondary sensors' algorithms to the MMA955xL's embedded microcontroller, users can develop unique features for their products.

The MMA955xL platform has Freescale-provided, pre-programmed applications and infrastructure modules such as a scheduler, filters, command interpreter, basic accelerometer, gesture recognition, and pedometery.

In addition, users can write their own custom applications adding to the software applications and infrastructure of the MMA955xL platform.

This document describes the functionality, configuration, and outputs of all the Freescale-provided applications.

2.2 MMA955xL package: axis orientation

The package orientation and measured values are shown in the following figure, PU = Portrait Up, PD = Portrait Down, LL = Landscape Left, LR = Landscape Right, Back = Back facing up, Front = Front facing up.

In the Portrait Up orientation, both X and Z axes read 0g and the Y axis reads the equivalent of -1g.

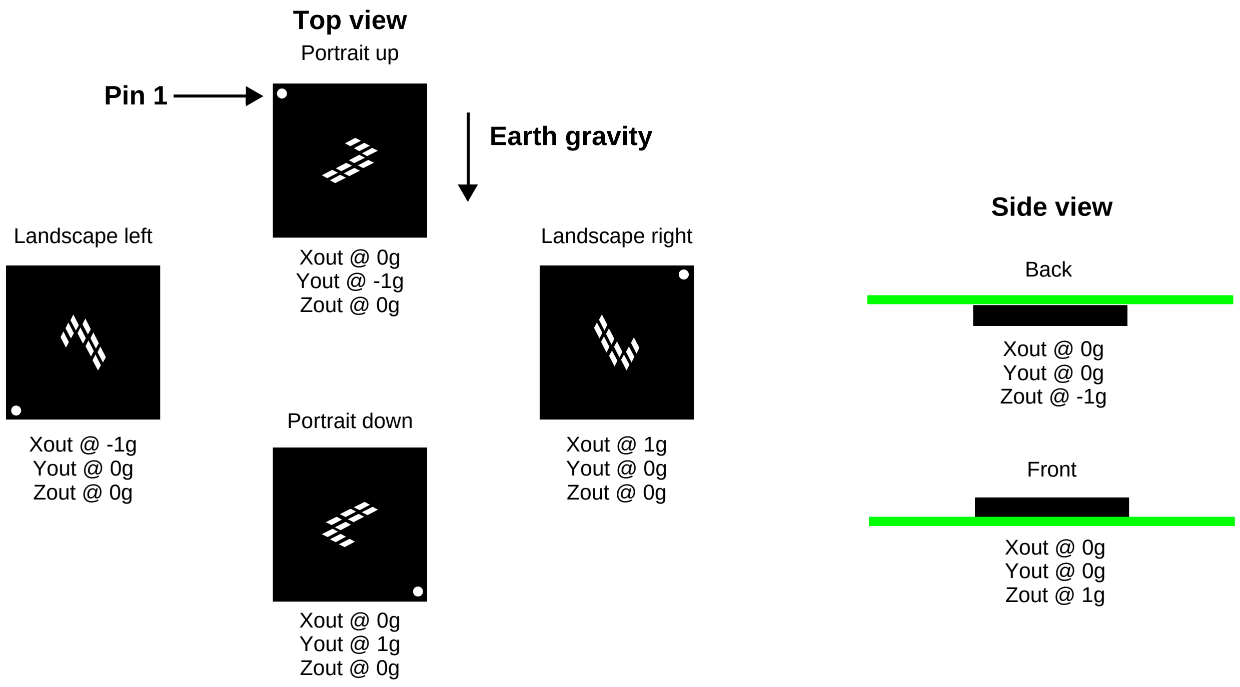


Figure 2-1. Device orientation

2.3 Data flow

The MMA955xL platform can act as a standalone, local processing unit or, more commonly, be connected to a host processor. The host processor configures and reads data from the MMA955xL platform.

The basic building blocks of the MMA955xL platforms are the MEMS sensor accelerometer, the analog-to-digital converters, signal conditioning, and an embedded ColdFire V1 microprocessor. The physical data path is relatively simple, as shown in the following figure.

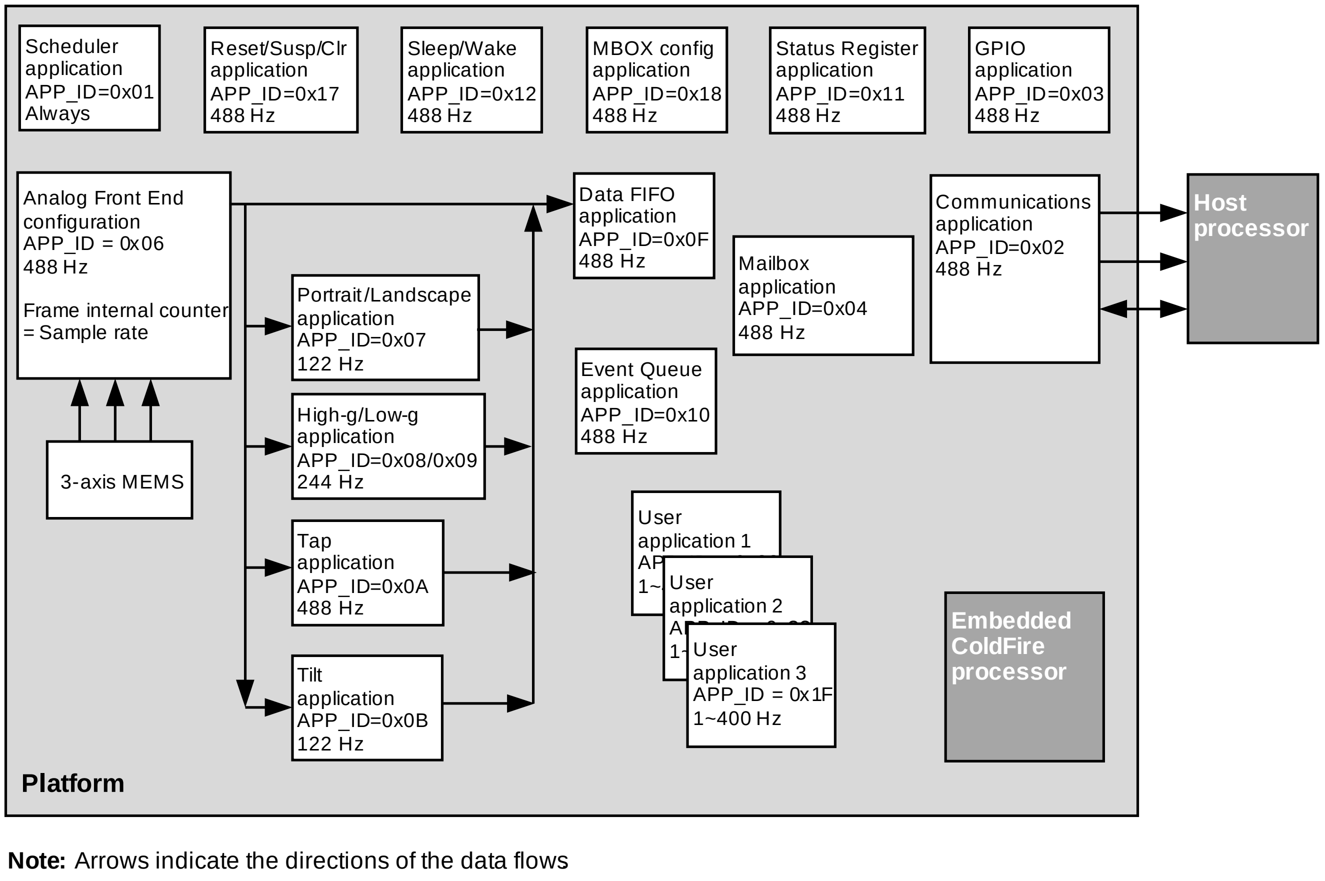

Scheduler application APP_ID=0x01 Always
Reset/Susp/Clr application APP_ID=0x17 488 Hz
Sleep/Wake application APP_ID=0x12 488 Hz
MBOX config application APP_ID=0x18 488 Hz
Status Register application APP_ID=0x11 488 Hz
GPIO application APP_ID=0x03 488 Hz
Analog Front End configuration APP_ID = 0x06 488 Hz
Frame internal counter = Sample rate
3-axis MEMS
Portrait/Landscape application APP_ID=0x07 122 Hz
High-g/Low-g application APP_ID=0x08/0x09 244 Hz
Tap application APP_ID=0x0A 488 Hz
Tilt application APP_ID=0x0B 122 Hz
Data FIFO application APP_ID=0x0F 488 Hz
Event Queue application APP_ID=0x10 488 Hz
Mailbox application APP_ID=0x04 488 Hz
Communications application APP_ID=0x02 488 Hz
Host processor
User application 1
User application 2
User application 3 APP_ID = 0x1F 1~400 Hz
Embedded ColdFire processor
Platform
Note: Arrows indicate the directions of the data flows

Figure 2-2. Platform data flow

Internal applications, that process the sensor data, run on the embedded ColdFire processor.

The functional flow of the MMA955xL platform includes the following:

1. A mechanical acceleration event causes a small mass to move inside the MEMS sensor.
2. The motion or offset is converted to a small voltage and amplified.
3. The analog-to-digital converter samples the voltage and converts it to a digital number.

   Those digital numbers are available for the embedded ColdFire processor to read and process.
4. Freescale-provided applications, running on the ColdFire processor, process the data and provide high-level analysis such as angle, position, and gesture recognition.
5. If desired, user applications implement extended or new functions and features.
6. If desired and programmed, the MMA955xL platform interrupts the host processor on specific conditions.
7. If desired and programmed, the host processor polls or responds to the MMA955xL- interrupts and collects the processed information from the MMA955xL platform.

2.4 User applications

User applications are a collection of functions grouped with a header structure. Such applications consist of:

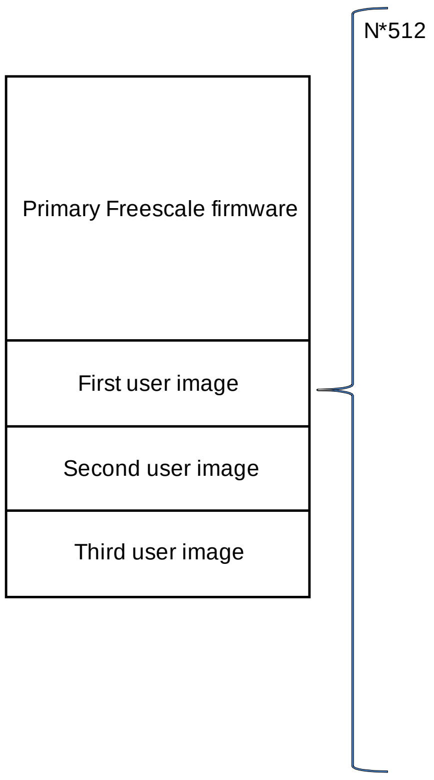
- A header structure for the scheduler to find the application and callback functions
- A set of callback functions for the scheduler to control the user application
- The application code

The header structure is called an application table. This table is a structure that contains the table identifier, the application identifier (APP_ID), the number of configuration and output registers, and pointers to the initialization, reset, clear, and application functions.

To enable the scheduler to find the applications at boot time, they must be linked and located at specific addresses in the flash memory. Multiple applications can be handled by the scheduler, provided there is sufficient space available in memory.

Users can implement up to three application tables. Each application table can contain many applications. For more details, see “User Applications” on page 197.

The following image shows the flash memory map with the Freescale and user memory sections:



```
    app_table_t app_table = {
       TABLE_IDENTIFIER,      // table identifier = 0x9559CODE
       1,                     //num_of_applications
    {
       {(cbFunction)(&initCbFunction),     // init function address
        (cbFunction)(&resetCbFunction),    // reset function address
        (cbFunction)(&clearCbFunction),    // clear function address
        (cbFunction) &UserApplication,     // main function address
        (uint8_t)(CUST_FBID),              // application id
        (7),                               // number of parameter bytes
        (1)                                // number of output bytes
     }
    }                               ,      // call back block for
                                              application 2,if used....
};

    void initCbFucntion(void )
{
    /*    Request memory RAM for User Application variables*/
    /*    Set priority and activity of UserApplication1*/
    /*    Initilize User Application variables*/

}

    void  resetCbFunction(void ){}

    void clearCbFunction(void){}

    void  UserApplication1(void )

{

}
```

Figure 2-3. Flash memory map

2.4.1 Application table

The MMA955xL platform supports user applications that are contained in up to three additional application tables. These additional applications must follow these rules:

- Valid application tables are located at the 512-byte, flash-memory page boundaries

  The 32-bit addresses for functions and data within this map can refer to any absolute, flash-memory, or RAM location. (For more information, see "RAM allocation".)

- The embedded ColdFire processor's register A5 cannot be read nor modified
- Each user application must have a unique application identifier (APP_ID).

  The user must not use any of the Freescale application identifiers. (For a list of these identifiers, see Table 2-1 on page 22.)

- For all static RAM variables, the memory-allocation API function should be used. All RAM allocation must be included in the initialization callback function.

  The initialization callback function provides a way to execute some initialization code before any application is run. (For further information, see "Memory Allocation for User Applications" on page 193.)

These rules enable the scheduler to determine the locations of user applications at runtime, execute those applications in the appropriate priority, and access the applications' configuration and output/status registers. From the host interface, Freescale applications and user applications are configured and read in the same way. Both are normal applications and are identified by their APP_ID.

The definition and structure of an application table is shown in the following example.

Example 2-1. Application table

```
typedef struct app_table_tag {
  uint32 table_identifier;        // magic code to identify app_table = 0x9550C0DE
  uint32 num_of_apps;             // number of entries in this app_table
  Data_APMap_t apmap[];           // actual app_table table
} app_table_t;

typedef struct Data_APMap_tag
{
  void (*initCbFn)(void);         // init callback function pointer
  void (*rstCbFn)(void);          // reset callback function pointer
  void (*clrCbFn)(void);          // clear callback function pointer
  void (*applicationMain)(void);  // Application function pointer
  uint8_t APP_ID;
  uint8_t parameter_bytes;        // Number of bytes in the configuration parameters
  uint8_t output_bytes;
}Data_APMap_t;
```

For information on creating your own custom application to run on the MMA955xL platform, see "User Applications" on page 197 and *Building Custom Applications on MMA9550L/MMA9551L* (AN4129). (To access the application note, see "References" on page 13.)

2.4.2 RAM allocation

An application may reserve a portion of the system RAM by calling the system API function `RequestDataRam` function, which has the following format:

```
  uint8_t * RequestDataRam(uint16 sz, uint8 u8app_id);
```

The required parameters are the size of the RAM to be reserved and the identifier of the application requesting RAM.

The function returns the address of the allocated variable.

Allocated memory is local to the application that requested the memory. Memory allocation can only be done once and should be done in the initialization function. There is no mechanism to free or return memory.

If there is not enough memory available, a NULL pointer is returned, indicating an error.

For more information on memory allocation, see "Memory Allocation for User Applications" on page 193.

2.5 System initialization

Two types of resets are recognized in the system: power on reset (POR) and pin reset (reset).

After a POR, the MMA955xL platform's bootloader reads device-specific trim codes (that were programmed at the factory) and other parameters from the flash memory to configure the hardware. The bootloader also copies some of this information into working RAM for use by the platform software. It is important to power cycle the device after programming the flash, so that the new values programmed into the flash are copied to the working RAM area.

After a pin reset, the microprocessor boots—depending on some system parameters—and starts the scheduler. The scheduler checks every 512-byte, flash-memory page boundary for valid application tables that are identified, at the start of the page, by the four-byte keyword `0x9550C0DE`.

The scheduler-initialization sequence then builds a list of applications from the application tables to quickly access the application properties during normal, runtime execution. Any applications set to the same priority are executed in the order defined by the application table. Application tables at lower addresses are processed before those at higher addresses.

It is up to the user to select App_IDs that do not conflict with Freescale applications or other user-provided applications. If an APP_ID is reassigned, unpredictable system behavior may result.

2.5.1 Application identifiers

Freescale provides a suite of preloaded applications that vary with each device member of the MMA955xL family. The following table shows those applications, their APP_IDs, and whether they can access the configuration and status registers. The table also shows whether the application is preloaded for a device and the application's run rate, if it is supported by the device.

There are 14 unused APP_IDs available for future Freescale applications as well as user applications

Table 2-1. Platform applications

Freescale application	APP_ID	X = Access to these registers		X = Preloaded		Run Rate (Hz)
		Configuration registers	Status registers	MMA9550L	MMA9551L	
Version	0x00	—	X	X	X	Always runs
Scheduler	0x01	X	X	X	X	Always runs
Communications	0x02	—	—	X	X	488
GPIO	0x03	X	—	X	X	488
Mailbox	0x04	X	—	X	X	488
Reserved (Do not use)	0x05	—	—	—	—	—
Analog Front End	0x06	X	X	X	X	488
Portrait Landscape	0x07	X	X	—	X	122
High-g Detection	0x08	X	X	—	X	244
Low-g Detection	0x09	X	X	—	X	244
Tap Detection	0x0A	X	X	—	X	488
Tilt	0x0B	X	X	—	X	122
(Available for user applications)	APP_ID = 0x0C and 0x0D					
Frame Counter	0x0E	X	X	X	X	488
Data FIFO	0x0F	X	X	X	X	488
Event Queue	0x10	X	X	X	X	488
Status Register	0x11	X	X	X	X	488
Sleep Wake	0x12	X	X	X	X	488
(Available for user applications)	APP_ID = 0x13, 0x14, 0x15, 0x16					
Reset Suspend Clear	0x17	X	—	X	X	488
MBOX Config	0x18	X	—	X	X	488
(Available for user applications)	APP_ID = 0x 19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F					

2.6 Registers summary

A summary of all the configuration and status/output registers, for the Freescale-provided applications, is given in the following table. For details on the specific register and status information, see the individual applications' chapters.

The Configuration registers hold parameters to configure and set up the applications. The Status registers show the output bits or data from an application.

Table 2-2. Freescale-provided applications' registers ([1])

Scheduler application (APP_ID = 0x01)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00-0x03	32	Request_to_start
	0x04-0x07	32	SFD Interrupt_AppIDs
	0x08-0x0B	32	AFE COCO Interrupt_AppIDs
	0x0C-0x0F	32	IRQ Interrupt_AppIDs
	0x10-0x13	32	TPMOV Interrupt_AppIDs
	0x14-0x17	32	TPMCH0 Interrupt_AppIDs
	0x18-0x1B	32	TPMCH1 Interrupt_AppIDs
	0x1C-0x1F	32	MTIMOV Interrupt_AppIDs
	0x20-0x23	32	PDBA Interrupt_AppIDs
	0x24-0x27	32	PDBB Interrupt_AppIDs
	0x28-0x2B	32	Master I^{2}C Interrupt_AppIDs
	0x2C	8	sched_parms_APP_ID_0x00
	0x2D - 0x4A	8 each	sched_parms_APP_ID_0x01 through 0x1E([1])
	0x4B	8	sched_parms_APP_ID_0x1F
Status	**Address**	**Width (bits)**	**Register Name**
	0x00-0x03	32	Timeout Status

Communications application (APP_ID = 0x02)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00	8	Config
Status	**Address**	**Width (bits)**	**Register Name**
	(None)		

1. Shaded rows indicate the compressed registers.

Table 2-2. Freescale-provided applications’ registers (continued)[1]

GPIO application (APP_ID = 0x03)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00	8	APP_ID for GPIO6
	0x01	8	SR Bit for GPIO6
	0x02	8	APP_ID for GPIO7
	0x03	8	SR Bit for GPIO7
	0x04	8	APP_ID for GPIO8
	0x05	8	SR Bit for GPIO8
	0x06	8	APP_ID for GPIO9
	0x07	8	SR Bit for GPIO9
	0x08-0x09	16	GPIO_pol; Polarity control bits
Status	**Address**	**Width (bits)**	**Register Name**
	(None)		

1. Shaded rows indicate the compressed registers.

Table 2-2. Freescale-provided applications' registers (continued)([1])

Mailbox application (APP_ID = 0x04)			
	Address	**Width (bits)**	**Register Name**
Configuration	0x00,0x01	16	Mailbox 4, APP_ID and Byte ID
	0x02,0x03	16	Mailbox 5, APP_ID and Byte ID
	0x04,0x05	16	Mailbox 6, APP_ID and Byte ID
	0x06,0x07	16	Mailbox 7, APP_ID and Byte ID
	0x08,0x09	16	Mailbox 8, APP_ID and Byte ID
	0x0A,0x0B	16	Mailbox 9, APP_ID and Byte ID
	0x0C,0x0D	16	Mailbox 10, APP_ID and Byte ID
	0x0E,0x0F	16	Mailbox 11, APP_ID and Byte ID
	0x10,0x11	16	Mailbox 12, APP_ID and Byte ID
	0x12,0x13	16	Mailbox 13, APP_ID and Byte ID
	0x14,0x15	16	Mailbox 14, APP_ID and Byte ID
	0x16,0x17	16	Mailbox 15, APP_ID and Byte ID
	0x18,0x19	16	Mailbox 16, APP_ID and Byte ID
	0x1A,0x1B	16	Mailbox 17, APP_ID and Byte ID
	0x1C,0x1D	16	Mailbox 18, APP_ID and Byte ID
	0x1E,0x1F	16	Mailbox 19, APP_ID and Byte ID
	0x20,0x21	16	Mailbox 20, APP_ID and Byte ID
	0x22,0x23	16	Mailbox 21, APP_ID and Byte ID
	0x24,0x25	16	Mailbox 22, APP_ID and Byte ID
	0x26,0x27	16	Mailbox 23, APP_ID and Byte ID
	0x28,0x29	16	Mailbox 24, APP_ID and Byte ID
	0x2A,0x2B	16	Mailbox 25, APP_ID and Byte ID
	0x2C,0x2D	16	Mailbox 26, APP_ID and Byte ID
	0x2E,0x2F	16	Mailbox 27, APP_ID and Byte ID
	0x30,0x31	16	Mailbox 28, APP_ID and Byte ID
	0x32,0x33	16	Mailbox 29, APP_ID and Byte ID
	0x34,0x35	16	Mailbox 30, APP_ID and Byte ID
	0x36,0x37	16	Mailbox 31, APP_ID and Byte ID

1. Shaded rows indicate the compressed registers.

Table 2-2. Freescale-provided applications' registers (continued)[1]

Status	Address	Width (bits)	Register Name
	(None)		

Analog Front End (AFE) application (APP_ID = 0x06)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00,0x01	16	AFE CSR
	0x02,0x03	16	User Offset X
	0x04,0x05	16	User Offset Y
	0x06,0x07	16	User Offset Z
	0x08	8	High Pass Filter Cutoff Coefficient
	0x09	8	Low Pass Filter Cutoff Coefficient
	0x0A	8	Temperature Low Pass Filter Cutoff Coefficient
	0x0B	8	EIC Low Pass Filter Cutoff Coefficient
	0x0C	8	Sample Rate Configuration
Status	**Address**	**Width (bits)**	**Register Name**
	0x00 - 0x05	48	Stage 0 – XYZ (16 bits each for X, Y, and Z)
	0x06 - 0x0B	48	Stage 1 – XYZ (16 bits each for X, Y, and Z)
	0x0C - 0x11	48	Stage 0 ABS – XYZ (16 bits each for X, Y, and Z)
	0x12 - 0x17	48	Stage 0 RAW – XYZ (16 bits each for X, Y, and Z)
	0x18 - 0x1D	48	Stage 0 LPF – XYZ (16 bits each for X, Y, and Z)
	0x1E - 0x23	48	Stage 0 HPF – XYZ (16 bits each for X, Y, and Z)
	0x24 - 0x25	16	Output Temperature
	0x26 - 0x27	16	Output EIC
	0x29 - 0x29	16	Frame Counter

1. Shaded rows indicate the compressed registers.

Table 2-2. Freescale-provided applications’ registers (continued)([1])

Portrait/Landscape application (APP_ID = 0x07)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00	8	Threshold_tilt
	0x01	8	Landscape_angle
	0x02	8	Portrait_angle
	0x03	8	Debounce_count
	0x04	8	Hysteresis _LO
	0x05	8	Hysteresis_BAFRO (Back to Front)
	0x06	8	Configuration
Status	**Address**	**Width (bits)**	**Register Name**
	0x00	8	PL_OUT

High-g/Low-g application (High g APP_ID = 0x08; Low g APP_ID = 0x09)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00,0x01	16	Low_g_thresh
	0x02	8	Low_g_cnt_min
	0x03	8	Low_g_config
	0x04,0x05	16	HI_g_thresh
	0x06	8	HI_g_cnt_min
	0x07	8	HI_g_config
	0x09	8	Lhg_event_mask
Status	**Address**	**Width (bits)**	**Register Name**
	0x00	8	Lhg_out

1. Shaded rows indicate the compressed registers.

Table 2-2. Freescale-provided applications' registers (continued)[1]

Tap Detection application (APP_ID = 0x0A)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00,0x01	16	tap_thresh
	0x02	8	Tap_on_min
	0x03	8	Tap_on_max
	0x04	8	TapTap MinTime
	0x05	8	Tap_K_HP
	0x06	8	Tap_K_LP
	0x07	8	Tap_Axis_enable
	0x08	8	Tap_events_mask
Status	**Address**	**Width (bits)**	**Register Name**
	0x00	8	Tap Out
	0x01	8	Double Tap Out

Tilt application (APP_ID = 0x0B)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00	8	Tilt K_LP
	0x01	8	Tilt Confg 1
	0x02	8	Tilt Event Mask
Status	**Address**	**Width (bits)**	**Register Name**
	0x00	8	Tilt Delta xz angle
	0x01	8	Tilt Delta yz angle
	0x02	8	Tilt Delta xy angle
	0x02	8	Tilt z xyz quad

1. Shaded rows indicate the compressed registers.

Table 2-2. Freescale-provided applications' registers (continued)[1]

Frame Counter application (APP_ID = 0x0E)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	(None)		
Status	**Address**	**Width (bits)**	**Register Name**
	0x00,0x01	16	Frame Count

Data FIFO application (APP_ID = 0x0F)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00	8	FIFO CONFIG
	0x01-0x03	Not used	
	0x04,0x05	16	FIFO SIZE WORD
	0x06	8	FIFO Channel APP_ID
	0x07	Not used	
	0x08,0x09	16	Watermark
Status	**Address**	**Width (bits)**	**Register Name**
	0x00,0x01	16	Records Number
	0x02	8	Entry Size
	0x03	8	FIFO STATUS

Event Queue application (APP_ID = 0x10)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00,0x01	16	Queue Size
	0x02,0x03	16	Queue Watermark
	0x04,0x05	16	Queue Timeout
Status	**Address**	**Width (bits)**	**Register Name**
	0x00,0x01	16	Records Number
	0x02	8	Entry Size
	0x03	8	QUEUE STATUS

1. Shaded rows indicate the compressed registers.

Table 2-2. Freescale-provided applications’ registers (continued)[1]

Status Register application (APP_ID = 0x11)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00, 0x01	16	Status Bit 0; APP_ID and SR BIT
	0x02 … 0x0D	16	Status Bits 1 through 6; APP_ID and SR BIT[(1)]
	0x0E,0x0F	16	Status Bit 7; APP_ID and SR BIT
Status	**Address**	**Width (bits)**	**Register Name**
	0x00,0x01	16	Status Register

Sleep / Wake application (APP_ID = 0x12)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00,0x01	16	Sensitivity threshold
	0x02,0x03	16	Doze Time Threshold
	0x04	8	Long Time Off
	0x05	8	Short Time Off
	0x06	8	Config
Status	**Address**	**Width (bits)**	**Register Name**
	0x00	8	Scheduler Mode

Reset / Suspend / Clear application (APP_ID = 0x17)			
Configuration	**Address**	**Width (bits)**	**Register Name**
	0x00-0x03	32	Reset Control Bits
	0x04-0x07	32	Suspend Control Bits
	0x08-0x0B	32	Clear Control Bits
Status	**Address**	**Width (bits)**	**Register Name**
	(None)		

1. Shaded rows indicate the compressed registers.

Chapter 3 Version Application

Application ID	0x00
Default speed	Always available.
Configuration registers	None.
Status registers	None.

The MMA955xL device's system-version information is stored in a 12 byte packet and contains system-identity information including device-hardware ID; the versions of the ROM bootloader, primary firmware, and hardware; and the system-build information.

The following table describes the system-version packet and its corresponding mailbox alignment.

Table 3-1. Version command description bytes

Mailbox number	Description	Byte
4	Device identifier	31:24
5	Device identifier	24:16
6	Device identifier	15:8
7	Device identifier	7:0
8	ROM major version number	7:0
9	ROM minor version number	7:0
10	Firmware major version number	7:0
11	Firmware minor version number	7:0
12	Hardware major version number	7:0
13	Hardware minor version number	7:0
14	Build major version number	7:0
15	Build minor version number	7:0

The following table describes the “Build Major” and “Build Minor” version-number fields.

Table 3-2. Version application, Build major and minor bytes

Byte	Address	Description	Bit fields
3	0x1FD	Build major version number	• 7–3 Build’s day of the month Range, 1 to 31 • 2–0 Year of build, from 2010 Range, 0 to 7.
4	0x1FC	Build minor version number	• 7 — Release – 1 Engineering version – 0 Production release • 6–4 Build number • 3–0 Month of build Range, 1 to 12

3.1 Reading the version information

To read the MMA955xL platform's version information, send the following command packets to the mailboxes:

1. MB0: Set APP_ID to 0x00.
2. MB1: Command to 0x00.
   Reads version information.
3. MB2: Set offset to zero 0x00.
   Starts reading at offset 0.
4. MB3: Set count field to 0x0C.
   Reads12 bytes of data.

Bytes to Send: 0x00, 0x00, 0x00, 0x0C.

The expected response to these commands is given in the following example.

Example 3-1.

```
00 80 0C 0C 2F 33 48 B8 01 01 02 02 01 06 03 41 00

MB0: APP_ID = 0x00
MB1: STATUS = 0x80 Command Complete, no errors
MB2: RequestedData count= 0xC
MB3: Actual Data Count= 0xC
MB4-7: Device ID = 0x2F3348B8
MB8-9: ROM Version = 01.01
MB10-11: Firmware Version = 02.02
MB12-13: Hardware Version = 01.06
MB14-15: Firmware Build = 03.41 (Production #3, 4 Mar 2011)
```

Chapter 4 Scheduler Application

Application ID	0x01
Default speed	(Runs all the time)
Configuration registers	Start on page 42.
Status registers	Start on page 42.

A simple task scheduler manages execution of the applications of the MMA955xL platform. Based on a run-to-completion scheme, the scheduler features very low cycle and memory overhead.

The scheduler is tightly coupled with the Analog Front End's sampling rate and is triggered at the start of every sample interval. The system is designed such that all applications must complete their processing within the sample interval.

The scheduler handles both Freescale and user applications.

A priority scheme allows short-duration, high-priority tasks. —such as data sampling and filtering—to preempt long-duration, low-priority tasks.

The scheduler scans serially through the list of applications, looking for applications that have the same priority as the scheduler's current priority. When there is a priority match, an activity level is checked to determine if the application should be run in the current scheduler's interval.

An application’s activity mask can be set to High, Low, Both, or None. This feature allows an application to run during high activity, low activity, both, or not at all. The Sleep/Wake application defines the thresholds of activity between Sleep and Wake, or High and Low activity.

Priorities from 16 to 23 are linked directly to the frame execution rate. (See Table 4-1 on page 35.) The lower-priority levels provide a range of values for managing applications in the user system.

The scheduler automatically executes all the applications with a priority corresponding to the scheduler's current running priority level.

Once the scheduler has identified an application to run, it does a context switch to that application. When the application completes, context is returned to the scheduler and it looks for more applications with the same priority.

4.1 Scheduler operational overview

The basic operational steps of the scheduler include:

1. The scheduler determines what type of interrupt occurred.

   If an application-ready interrupt occurred (such as afe_results_ready or mtim_overflow), that application's bit is set in the request_to_start register and the interrupt is acknowledged to the hardware.

   If an application-complete interrupt occurred (because an application returned to the invoke_scheduler function), the application's bit is cleared in the started register and context is popped from the user stack.
2. The highest priority started or request_to_start application is determined.

   If this application is not started, the context of the old application is pushed onto the user stack.

   If the application is waiting to start, a context switch pushes the embedded ColdFire processor’s registers D0-D2/A0-A1/SR/PC, MACSR, and ACC onto the ColdFire processor's user stack. It also pushes the address of the internal function `invoke_scheduler()` onto the user stack—which causes the program flow to return to the scheduler when the application completes.
3. The context switch modifies the ColdFire processor’s supervisor stack to redirect the interrupt to return to the application in user mode.

   This clears the request_to_start bit and sets the start bit for the application. If the application is already started, no context push is required.
4. The interrupt handler automatically returns to the application.
5. When the scheduler finds no more applications at the current priority level, it searches for the next-lower priority level.
6. When all priorities have been run, the scheduler enters an idle state.

4.2 Scheduler application elements

MMA955xL applications consist of the application code, a set of specifically defined callback functions that the scheduler uses to control the application, and a header structure that enables the scheduler to find the application and callback functions. For the scheduler to find the applications at boot time, they must be linked and located at specific addresses in the memory.

Users can implement up to three application tables. Each application table can include multiple applications.

Applications are assigned a priority and an activity level. These two parameters determine when the scheduler runs the application.

For more information on applications, see “User Applications” on page 197.

4.2.1 Priority levels

A priority scheme allows short-duration, high-priority applications—such as data sampling and filtering—to preempt long-duration, low-priority applications.

The scheduler supports 24 unique priorities, from 0x00 to 0x17 (0 to 23, in decimal). The larger the number, the higher the priority—with such applications run before applications with lower numbers. Multiple applications can be assigned the same priority level.

The priority level for each application is encoded by the lower six bits in the scheduler_parameters registers. See “Scheduler parameters register” on page 50.

Priorities from 0 to 15 (0x00 to 0x0F) can trigger an application when an interrupt occurs. Priorities 16 to 23 (0x10 to 0x17) are linked to the Analog Front End (AFE) execution rate. All applications with a priority of 16 to 23 are automatically software-triggered by the Scheduler.

Table 4-1. Priorities descriptions

Priority Level	Description
0x17	Applications with this priority run at 488 Hz
0x16	Applications with this priority run at 244 Hz
0x15	Applications with this priority run at 122 Hz
0x14	Applications with this priority run at 61 Hz
0x13	Applications with this priority run at 30 Hz
0x12	Applications with this priority run at 15 Hz
0x11	Applications with this priority run at 7 Hz
0x10	Applications with this priority run at 3 Hz
0x0F	Highest user assignable priority level (lower than 3 Hz)

Table 4-1. Priorities descriptions (continued)

Priority Level	Description
0x0E	User-assignable priority levels
0x0D	
0x0C	
0x0B	
0x0A	
0x09	
0x08	
0x07	
0x06	
0x05	
0x04	
0x03	
0x02	
0x01	
0x00	Lowest user assignable priority level

4.2.2 Activity levels

The MMA955xL platform uses an “activity level” attribute to indicate how physically active the device is. The measured physical acceleration and motion actions of the device are classified into two ranges: high activity and low activity.

The activity level is used with the priority level to determine which applications run first. Applications can be set to run during these conditions:

- Always run
- Run during high activity
- Run during low activity
- Never run

The activity level for each application is encoded in the upper two bits of the scheduler_parameters registers. See “Scheduler parameters register” on page 50.

The Sleep/Wake application alone determines the activity level, therefore high/low activity is entirely a function of the sleep/wake threshold parameters. When in run mode, the scheduler only executes applications with the high activity parameter bit set. Similarly, when in doze mode, the scheduler only executes applications with the low activity parameter bit set. Since these bits are separate, an application has the option to set both to run regardless of the activity level.

The low activity level parameter is like a filter which skips applications that would normally run after each AFE sample interval. The idea is that when the sensor is not moving (i.e., sitting on your desk), you can save power by not running applications like portrait landscape.

4.3 Interrupts

Interrupts immediately divert execution from any application or the idle state into dedicated interrupt handlers. The scheduler is not invoked, so the interrupt returns without redirection.

All interrupt handlers disable interrupts, so application ready, communication, and pin interrupts divert execution only from applications or the idle state.

Interrupts are never nested.

User interrupt handlers should be made as small and as fast as possible.

The MMA955xL platform supports the interrupts listed in the following table. Each supported interrupt can be associated with zero or more priorities. When the interrupt occurs, the scheduler triggers all the applications associated with the priorities configured for that interrupt.

The priorities for each interrupt are configured in the scheduler configuration parameter structure.

Table 4-2. Supported interrupts (1)

Name	Description
SFD	Start of Digital Frame interrupt. This interrupt occurs when the start of frame signal is asserted.
COCO	Conversion-complete interrupt. This interrupt occurs when the AFE cycle has completed and all ADC conversions are complete and ready to be used.
IRQ	IRQ pin interrupt. This interrupt occurs when the selected input pin detects the configured condition (rising edge/high level or falling edge/low level).
TPMOV	Timer overflow interrupt. This interrupt occurs when the TPM counter resets to 0x00 after reaching the modulo value programmed in the TPM counter modulo register.
TPMCH0	Timer channel 0 interrupt. This interrupt occurs when the value in the TPM counter matches the value in the TPM channel 0 value register.
TPMCH1	Timer channel 0 interrupt. This interrupt occurs when the value in the TPM counter matches the value in the TPM channel 0 value register.
MTIMOV	MTIM16 overflow flag. This interrupt occurs when the MTIM16 counter overflows to 0x0000 after reaching the value in the MTIM16 modulo register.
PDBA	Programmable delay block interrupt. This interrupt occurs after the trigger source has activated the PDB counter and the counter has matched the PBD modulo value.

Table 4-2. Supported interrupts (continued)[1]

Name	Description
PDBB	Programmable-delay block interrupt. This interrupt occurs after the trigger source has activated the PDB counter and the counter has matched the PBD modulo value.
MASTER_I2C	Inter-Integrated Circuit interrupt. This interrupt is linked to the I^{2}C master communication tasks.

1. For more information, see the *MMA955xL Intelligent, Motion-Sensing Platform Hardware Reference Manual* (MMA955xLRM), accessible from a link in “References” on page 13.

Interrupts are assigned to an application in the Interrupt Assignment registers. See “Interrupt assignment registers” on page 45.

4.4 Scheduler preemption

This section gives some detailed examples of how preemption is handled by MMA955xL platforms.

4.4.1 High-priority task suspending low-priority task

The scheduler has the ability to switch to a higher-priority application while a lower-priority application is running. This example examines a low-priority, counter application—such as for a pedometer—and a high-priority, data-processing application—that might be used for a special user task.

Example 4-1.

Initially, the user:

1. Configures the timer interrupt to point to the counter application's APP_ID, in the user_interrupt_register of the scheduler.
2. Sets the priority of the counter application—in the sched_parms register of the scheduler—to something low.
3. Sets the priority of the data-processing application—in the sched_parms register of the scheduler—to TASK488 or something higher than the counter application.

If the scheduler is in the idle state and all high- and low-priority applications have been run, the scheduler has nothing left to do and is waiting for an interrupt. In this example, the hardware timer counter has just overflowed and triggered an interrupt.

In this scenario, the MMA955xL platform responds as follows:

1. The interrupt handler acknowledges the interrupt to the hardware by clearing a flag in an MTIM register and schedules the counter application to run by setting the APP_ID bit in the request_to_start register.
2. The scheduler determines that the counter application is the highest priority executable application, pushes the idle state context onto the stack and calls the counter application.
3. While the counter application is still running, the Analog Front End (AFE) results become available—which triggers another interrupt.
4. The interrupt handler acknowledges the interrupt to the hardware and schedules the data-processing application to run by setting the data-processing application’s bit in the request_to_start register.
5. The scheduler determines that the data-processing application is the highest-priority executable task, pushes the counter application context and calls the data-processing application task to sample the ADC, trim, and filter.
6. Sometime later, the data-processing application finishes and returns to the scheduler via the `invoke_scheduler()` call.
7. The scheduler clears the data-processing application-started flag, pops the counter application context from the stack, and returns to the counter application to resume the pedometer application.
8. The counter application completes and returns to the scheduler.

9. The scheduler clears the counter application-started flag and pops the idle state context from the stack and returns to the idle state.

4.4.2 Low-priority task becoming ready during high-priority task

This example shows how the scheduler functions when a low-priority task becomes ready while a high-priority task is being executed. This operation uses three tasks: idle, MTIMOV, and SFD.

- Idle: A low-power wait state
- MTIMOV: A low-priority used by the pedometer application and triggered by the MTIM counter overflow
- SFD: A high-priority used to sample the ADC, trim, and filter

Example 4-2.

In this scenario, the MMA955xL platform behaves as follows:

1. Starting in the idle task, AFE results become ready and trigger an SFD interrupt.
2. The scheduler acknowledges the SFD interrupt to the hardware and marks the SFD task as request_to_start.
3. The scheduler determines the SFD task is the highest-priority executable task, pushes the idle task context onto the stack, clears the SFD request_to_start flag, sets the SFD started flag, and returns to the SFD task to sample the ADC, trim, and filter.
4. Before the SFD task completes, an MTIMOV interrupt occurs.
5. The scheduler acknowledges the interrupt to the hardware by clearing a flag in an MTIM register and marks the MTIM task as request_to_start.
6. The scheduler determines the SFD task is still the highest priority executable task and returns to the SFD task without any context switch.
7. Later, the SFD task completes and returns to the scheduler via a trap call.
8. The scheduler clears the SFD task-started flag and pops the idle task context from the stack.
9. The scheduler must start the MTIMOV task, so it pushes the idle task context back onto the stack, clears the MTIMOV request_to_start flag, sets its started flag, and returns into the MTIMOV task.

   Though it is not necessary to pop, then push the idle task context, this technique simplifies the scheduler logic—reducing the required code size.
10. The MTIMOV task completes and returns to the scheduler.
11. The scheduler clears the MTIMOV task-started flag, pops the idle task context from the stack, and returns to the idle task.

4.5 Error conditions

In the event that an application does not complete before it is triggered again, the first instance of the application runs to completion before the second instance starts. If a third trigger occurs before the first instance of the application completes, the application misses an instance and is marked in the timeout register.

By checking the timeout_status register, the user can determine which priority application was missed. The corresponding bit of the timeout will be set in the timeout_status register.

For more information on the timeout_status register, see Chapter 16, “Status Register Application” on page 161.

4.6 Scheduler configuration registers

The scheduler has the following three groups of configuration registers:

- request_to_start register
- interrupt assignment registers
- scheduler parameters registers

The configuration and status registers are listed in the following table.

Table 4-3. Scheduler configuration and status registers

Register type	Address	Width (bits)	Register Name
Configuration	0x00-0x03	32	Request_to_start
	0x04-0x07	32	SFD Interrupt_AppIDs
	0x08-0x0B	32	AFE COCO Interrupt_AppIDs
	0x0C-0x0F	32	IRQ Interrupt_AppIDs
	0x10-0x13	32	TPMOV Interrupt_AppIDs
	0x14-0x17	32	TPMCH0 Interrupt_AppIDs
	0x18-0x1B	32	TPMCH1 Interrupt_AppIDs
	0x1C-0x1F	32	MTIMOV Interrupt_AppIDs
	0x20-0x23	32	PDBA Interrupt_AppIDs
	0x24-0x27	32	PDBB Interrupt_AppIDs
	0x28-0x2B	32	Master I^{2}C Interrupt_AppIDs
	0x2C	8	sched_parms_APP_ID_0x00
	0x2D - 0x4A	8 each	sched_parms_APP_ID_0x01 through 0x1E(1)
	0x4B	8	sched_parms_APP_ID_0x1F
Status	**Address**	**Width (bits)**	**Register Name**
	0x00-0x03	32	Timeout Status

1. Shaded rows indicate the compressed registers.

The request_to_start register is a way to run the applications under scheduler control. The interrupt assignment registers connect applications to interrupt events. The scheduler parameter registers assign activity levels and priorities to the applications.

4.6.1 request_to_start register

An application can be scheduled to run by setting the bit corresponding to the application's APP_ID in the request_to_start register. The scheduler manages starting applications, depending on priority and activity.

User applications also may schedule other applications to run by setting the appropriate bit in the request_to_start register. When an application finishes and returns, the scheduler clears the corresponding bit in the request_to_start register.

The request_to_start register enables a user to trigger one or more applications via the slave-port command or a direct write-through from a user application. When modifying this register, the user must logically OR the current value with the new value (with a read/modify/write access) to prevent the erasure of the task waiting to be started.

Table 4-4. request_to_start register

Offset	0x00(MSB)								0x01							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	request_to_start [31:24]								request_to_start [23:16]							
Reset	0x00								0x00							
Offset	0x02								0x03(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	request_to_start [15:8]								request_to_start [7:0]							
Reset	0x00								0x00							

Table 4-5. request_to_start description

Field	Description
request_to_start [31:0]	Indicates which APP_IDs are marked for starting at the beginning of the next scheduler interval. Each bit corresponds to an APP_ID. It is important that the user do a read/modify/write type of access to ensure that the contents of the resister are logically ORed with the new value to avoid erasing the request_to_start bit of the other APP_IDs that are waiting to start. Setting the bit will mark the APP_ID for starting in the next scheduler loop. Units: None. Range of valid values: 0 to 0xFFFFFFF.

The host controller may issue a command through the slave communications port to trigger the running of an application. To do this, the host controller does a read/modify/write to the request_to_start configuration register and sets the appropriate bit associated with the application that the host wants to start.

Similarly, a user application running natively on the MMA955xL platform can trigger one or more applications to start by writing directly to the request_to_start register.

Since the scheduler only samples this configuration register through its interrupt handler, a trap interrupt is available for user code to invoke the scheduler. This mechanism is intended to support user-defined, software-triggered applications. Be advised, however, that there are no protections to prevent a slave-port command or a direct-write from triggering a hardware-assigned applications such as SFD or MTIMOV.

Regardless of which mechanism triggers the Scheduler application to start—a hardware interrupt, a slave port command, or a trap interrupt—it always executes the highest-priority applications first. As a result, a slave-port command could trigger a low-priority application, but some time may occur before that application actually executes.

4.6.2 request_to_start register configuration example

To configure the request_to_start register, send this command, read the register and modify it, and write it back.

Example 4-3.

1. MB0: Set the Scheduler application identifier (0x01).
2. MB1: Set the Command: Read Config application identifier (0x10).
3. MB2: Set the Offset to Zero field (0x00) to point to the request_to_start configuration register.
4. MB3: Set the Count field (0x04) to request four bytes.

Bytes to send: 0x01, 0x10, 0x00, 0x04.

5. Read back the mailboxes.

   The current value of the request_to_start register is stored in MB4-MB7.
6. Modify the value as needed:
7. MB0: Set the Scheduler application identifier (0x01).
8. MB1: Set the Command: Write Config application identifier (0x20).
9. MB2: Set the Offset to Zero field (0x00) to point to the request_to_start configuration register.
10. MB3: Set the Count field (0x04) to writing four bytes.
11. MB4: Set the DATA value to 0xAA.
12. MB4: Set the DATA value to 0xBB.
13. MB4: Set the DATA value to 0xCC.
14. MB4: Set the DATA value to 0xDD.

Bytes to send: 0x01, 0x20, 0x00, 0x04, 0xAA, 0xBB, 0xCC, 0xDD.

4.6.3 Interrupt assignment registers

There are 10 possible interrupts in the MMA955xL platform. Each interrupt is assigned to an application through the user_interrupt parameter registers. When an interrupt happens, the scheduler handler logically ORs the contents of the user_interrupt register with the request_to_start parameter register.

The scheduler runs the appropriate application the next time the application's priority is runable.

The scheduler uses the interrupt vector to determine which application-ready interrupt occurred and sets a bit in the request_to_start register. In the bit vector, each bit corresponds to a task and the bit position indicates the priority of the application.

The appropriate interrupt is acknowledged to the hardware, usually by clearing a flag bit in the peripheral's memory-mapped status register. In the special case that an AFE, results-ready interrupt occurred, supervisor-only AFE registers are copied into user-mode shadow registers.

Table 4-6. user_interrupt registers

Offset	0x04(MSB)								0x05							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	SFD Interrupt_AppIDs [31:24]								SFD Interrupt_AppIDs [23:16]							
Reset	0x00								0x00							
Offset	0x06								0x07(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	SFD Interrupt_AppIDs [15:8]								SFD Interrupt_AppIDs [7:0]							
Reset	0x00								0x00							

Offset	0x08(MSB)								0x09							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	AFE_COCO Interrupt_AppIDs [31:24								AFE_COCO Interrupt_AppIDs [23:16]							
Reset	0x00								0x00							
Offset	0x0A								0x0B(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	AFE_COCO Interrupt_AppIDs [15:8]								AFE_COCO Interrupt_AppIDs [7:0]							
Reset	0x00								0x00							

Table 4-6. user_interrupt registers (continued)

Offset	0x0C(MSB)								0x0D							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	IRQ Interrupt_AppIDs [31:24]								IRQ Interrupt_AppIDs [23:16]							
Reset	0x00								0x00							
Offset	0x0E								0x0F(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	IRQ Interrupt_AppIDs [15:8]								IRQ Interrupt_AppIDs [7:0]							
Reset	0x00								0x00							

Offset	0x10(MSB)								0x11							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	TPMOV Interrupt_AppIDs [31:24]								TPMOV Interrupt_AppIDs [23:16]							
Reset	0x00								0x00							
Offset	0x12								0x13(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	TPMOV Interrupt_AppIDs [15:8]								TPMOV Interrupt_AppIDs [7:0]							
Reset	0x00								0x00							

Offset	0x14(MSB)								0x15							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	TPMCH0 Interrupt_AppIDs [31:24]								TPMCH0 Interrupt_AppIDs [23:16]							
Reset	0x00								0x00							
Offset	0x16								0x17(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	TPMCH0 Interrupt_AppIDs [15:8]								TPMCH0 Interrupt_AppIDs [7:0]							
Reset	0x00								0x00							

Table 4-6. user_interrupt registers (continued)

Offset	0x18(MSB)								0x19							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	TPMCH1 Interrupt_AppIDs [31:24]								TPMCH1 Interrupt_AppIDs [23:16]							
Reset	0x00								0x00							
Offset	0x1A								0x1B(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	TPMCH1 Interrupt_AppIDs [15:8]								TPMCH1 Interrupt_AppIDs [7:0]							
Reset	0x00								0x00							

Offset	0x1C(MSB)								0x1D							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	MTIMOV Interrupt_AppIDs [31:24]								MTIMOV Interrupt_AppIDs [23:16]							
Reset	0x00								0x00							
Offset	0x1E								0x1F(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	MTIMOV Interrupt_AppIDs								MTIMOV Interrupt_AppIDs [7:0]							
Reset	0x00 [15:8]								0x00							

Offset	0x20(MSB)								0x21							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	PDBA Interrupt_AppIDs [31:24]								PDBA Interrupt_AppIDs [23:16]							
Reset	0x00								0x00							
Offset	0x22								0x23LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	PDBA Interrupt_AppIDs [15:8]								PDBA Interrupt_AppIDs [7:0]							
Reset	0x00								0x00							

Table 4-6. user_interrupt registers (continued)

Offset	0x24(MSB)								0x25							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	PDBB Interrupt_AppIDs [31:24]								PDBB Interrupt_AppIDs [23:16]							
Reset	0x00								0x00							
Offset	0x26								0x27(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	PDBB Interrupt_AppIDs [15:8]								PDBB Interrupt_AppIDs [7:0]							
Reset	0x00								0x00							

Offset	0x28(MSB)								0x29							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	MASTER_IIC Interrupt_AppIDs [31:24]								MASTER_IIC Interrupt_AppIDs [23:16]							
Reset	0x00								0x00							
Offset	0x2A								0x2B(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	MASTER_IIC Interrupt_AppIDs [15:8]								MASTER_IIC Interrupt_AppIDs [7:0]							
Reset	0x00								0x00							

Table 4-7. user_interrupt bit descriptions

Field	Description
Interrupt_AppIDs [31:0]	Bit vector indicating which tasks are to start after the interrupt. Each bit corresponds to a task and the bit position indicates the task's priority. This 32-bit parameter should match the priority of the task that is executing when the interrupt occurs. Units: None. Range of valid values: 0 to 0xFFFFFF.

4.6.3.1 Interrupt_App_IDs register configuration example

In this example, the user has an external interrupt source that wants to run a user application assigned to APP_ID = 0x19. This requires setting the 0x19 bit position in the IRQ_interrupt_APP_IDs register at offset 0x0C-0x0F.

Example 4-4.

1. MB0: Set the Scheduler application identifier (0x01).
2. MB1: Set the “Command: Write Config” application identifier (0x20).
3. MB2: Set the Offset to Zero field (0x0C) to point to the IRQ_interrupt_APP_IDs register.
4. MB3: Set the Count field (0x04) to write four bytes.
5. MB4: Set the DATA value to 0x20.

   This bit is the 0x19th bit in this 32 bit register.
6. MB4: Set the DATA value to 0x00.
7. MB4: Set the DATA value to 0x00.
8. MB4: Set the DATA value to 0x00.

Command to send for write: 0x01, 0x20, 0x0C, 0x04, 0x20, 0x00, 0x00, 0x00.

4.6.4 Scheduler parameters register

Both the Freescale and user applications have the same structure for their sched_parms registers. That structure is shown in the following figure. For more information on the (Activity) and (Priority) bits, see Table 4-9 on page 50.

Table 4-8. sched_parms register structure

Bit	7	6	5	4	3	2	1	0
Field	(Activity)		(Priority)					

Table 4-9. sched_parms_APP_ID bit descriptions

Field	Description
(Activity) (7:6)	Scheduler parameters for each task, this value allows accelerometer activity (motion) to determine whether a task should run. High and low activity thresholds are defined with the Sleep/Wake application. (See “Sleep/Wake Application” on page 169.) Units: None. Range of valid values: • 0x03: ALWAYS // execute application during high and low activity • 0x02: ACTIVE // execute application only during high activity • 0x01: INACTIVE // execute application only during low activity • 0x00: NEVER // never execute application
priority (5:0)	Scheduler parameters for each task, this value determine the priority of the application. Units: None. Range of valid values: • 0x17: TASK488HZ // task running at 488 Hz • 0x16:TASK244HZ // task running at 244 Hz • 0x15: TASK122HZ // task running at 122 Hz • 0x14: TASK61HZ // task running at 61 Hz • 0x13: TASK30HZ // task running at 30 Hz • 0x12: TASK15HZ // task running at 15 Hz • 0x11: TASK7HZ // task running at 7 Hz • 0x10: TASK3HZ // task running at 3 Hz • 0x0F: PRIORITY15 • 0x0E: PRIORITY14 • 0x0D: PRIORITY13 • 0x0C: PRIORITY12 • 0x0B: PRIORITY11 • 0x0A: PRIORITY10 • 9: PRIORITY9 • 8: PRIORITY8 • 7: PRIORITY7 • 6: PRIORITY6 • 5: PRIORITY5 • 4: PRIORITY4 • 3: PRIORITY3 • 2: PRIORITY2 • 1: PRIORITY1 • 0: PRIORITY0

The variable data for the sched_parms registers includes the register name, offset, and reset. Each of the possible application IDs (0x00–0x1F) is associated with a scheduler parameter register. The following table shows the APP_IDs and their associated register offsets and reset values.

Table 4-10. sched_parms registers' differentiating values

Register			Resets	
Offset	**Name**	**Application**	**Bits 7:6**	**Bits 5:0**
0x00		Version		
0x2C	sched_parms_APP_ID_0x00	Scheduler	0x00	0x00
0x2D	sched_parms_APP_ID_0x01		0x00	0x17
0x2E	sched_parms_APP_ID_0x02	Communications	0x03	0xD7
0x2F	sched_parms_APP_ID_0x03	GPIO	0x03	0xD7
0x30	sched_parms_APP_ID_0x04	Mailbox	0x03	0x17
0x31	sched_parms_APP_ID_0x05	Reserved	—	—
0x32	sched_parms_APP_ID_0x06	AFE	0x03	0xD7
0x33	sched_parms_APP_ID_0x07	Portrait/Landscape Detection	0x00	0x00
0x34	sched_parms_APP_ID_0x08	High-g Detection	0x00	0x00
0x35	sched_parms_APP_ID_0x09	Low-g Detection	0x00	0x00
0x36	sched_parms_APP_ID_0x0A	Tap Detection	0x00	0x00
0x37	sched_parms_APP_ID_0x0B	Tilt	0x00	0x00
0x38	sched_parms_APP_ID_0x0C	(User application)	0x00	0x00
0x39	sched_parms_APP_ID_0x0D	(User application)	0x00	0x00
0x3A	sched_parms_APP_ID_0x0E	Frame Counter	0x00	0x00
0x3B	sched_parms_APP_ID_0x0F	Data FIFO	0x03	0xD7
0x3C	sched_parms_APP_ID_0x10	Event Queue	0x03	0xD7
0x3D	sched_parms_APP_ID_0x11	Status Register	0x03	0xD7
0x3E	sched_parms_APP_ID_0x12	Wake/Sleep	0x03	0xD7
0x3F	sched_parms_APP_ID_0x13	(User application)	0x00	0x00
0x40	sched_parms_APP_ID_0x14	(User application)	0x00	0x00
0x41	sched_parms_APP_ID_0x15	(User application)	0x00	0x00
0x42	sched_parms_APP_ID_0x16	(User application)	0x00	0x00
0x43	sched_parms_APP_ID_0x17	Reset/Suspend/Clear	0x03	0xD7
0x44	sched_parms_APP_ID_0x18	Mailbox Configuration	0x00	0x00

Table 4-10. sched_parms registers' differentiating values (continued)

Register			Resets	
Offset	**Name**	**Application**	**Bits 7:6**	**Bits 5:0**
0x45	sched_parms_APP_ID_0x19	(User application)	0x00	0x00
0x46	sched_parms_APP_ID_0x1A	(User application)	0x00	0x00
0x47	sched_parms_APP_ID_0x1B	(User application)	0x00	0x00
0x48	sched_parms_APP_ID_0x1C	(User application)	0x00	0x00
0x49	sched_parms_APP_ID_0x1D	(User application)	0x00	0x00
0x4A	sched_parms_APP_ID_0x1E	(User application)	0x00	0x00
0x4B	sched_parms_APP_ID_0x1F	(User application)	0x00	0x00

4.7 Scheduler status registers

The scheduler applications contain a set of output or status registers. Status registers can be read by a host processor, through the I^2C/SPI slave-port read-data commands, or by internal applications, through direct reads.

The scheduler status is a 32-bit register that gives the priority levels for applications that have timed out. Applications time out when their priority level is the one that is currently running and they have been marked with “request_to_start.” Such an application has not finished running before it needs to start again.

4.7.1 Timeouts

A timeout condition indicates that the scheduler has been somehow compromised by a user application.

Table 4-11. Scheduler status register

Offset	0x00(MSB)								0x01							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	timeout_status[31:24]								timeout_status[23:16]							
Reset	0x00								0x00							
Offset	0x02								0x03(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	timeout_status[16:8]								timeout_status[7:0]							
Reset	0x00								0x00							

Table 4-12. sched_parms_APP_ID bit descriptions

Field	Description
timeout_status [31:0]	Indicates the priority level of the task that has timed out, one or more times. The register sets the corresponding priority-level bit when a task is timed out (currently running and being marked “request_to_start”). This bit must be monitored to verify that no user task is compromising the scheduler execution. Units: None. Range of valid values: 0 to 0xFFFFFFFF.

Chapter 5 Communication Interface

5.1 Overview of Communication Interface

All access to the MMA955xL platform is made via the slave, serial Communication Interface that is part of the hardware and firmware infrastructure of the platform.

Commands are sent from the host and through the slave communications port (either SPI or I^{2}C). The Communication Interface interprets the command and sends the data to the correct application. It also responds with error codes when appropriate.

The Communications Interface works with the Mailbox application to implement the command and response. The mailboxes' functionality is configured with two applications: the MBOX Configuration application (APP_ID = 0x18) and the MBOX application (APP_ID = 0x04).

5.2 Mailbox interface

Commands are received through a set of 32 mailboxes that are arranged consecutively to provide addressable memory regions. Each mailbox can hold one byte of data.

After a command has completed, the Communication Interface writes the results to the mailboxes and the results (response out) are retrieved by the host via the SPI or the I^{2}C slave interface.

The following figure shows the structure of the data packet when writing one byte into a specific mailbox.

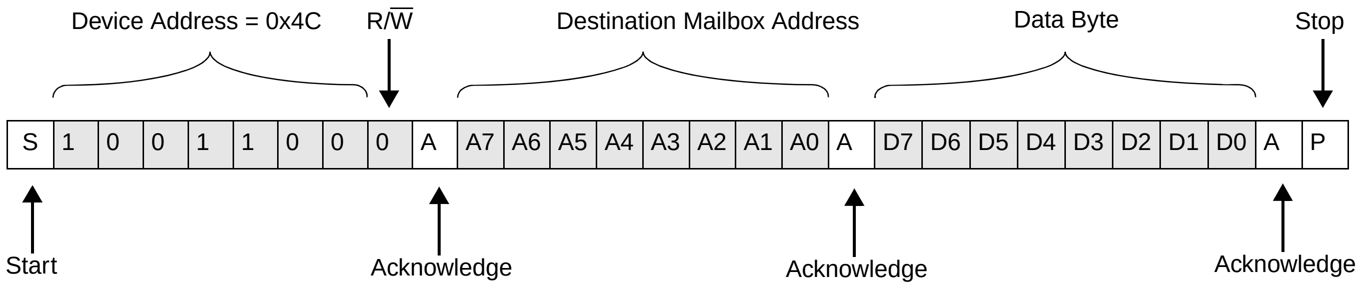


Figure 5-1. I^{2}C interface writing one byte of information to a mailbox

If the transaction contains more than one data byte, the internal-destination mailbox address is automatically incremented so that the incoming byte is placed in the next mailbox. For mailbox addresses greater than 31 bytes or for transactions where the mailbox address auto-increments past mailbox 31, the destination address wraps back to the start of the mailboxes addresses.

5.2.1 Mailbox timing diagrams

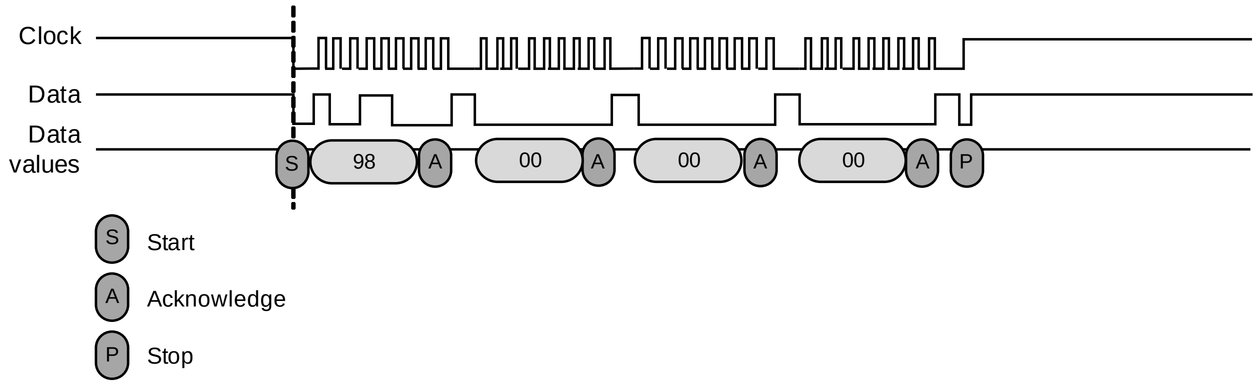
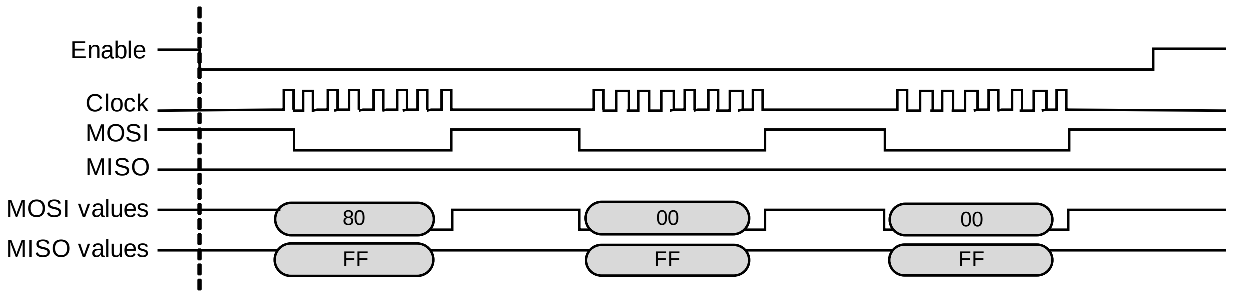


Figure 5-2. I^{2}C timing diagram



MISO: SPI Master In, Slave Out
MOSI: SPI Master Out, Slave In
SPI: Serial, Peripheral Interface

Figure 5-3. SPI timing diagram

5.3 Mailbox usage

Commands to the MMA955xL platform consist of a write followed by one or more reads. It may take some time to complete the command and a flag can be checked to determine if the command has completed. That flag is the Command Complete (COCO) bit, the seventh bit of the read data in the second mailbox. (See Table 5-4 on page 60.)

On a read operation, the COCO bit indicates if the command has been processed. The host processor can determine the status of the command's processing by repeatedly reading or polling the second mailbox until the COCO bit is set. Alternatively, the MMA955xL platform can be configured to assert an interrupt signal at the completion of a command. If configured, the INT_O interrupt will be set immediately after the COCO bit has been set.

For more information, see "Configuring mailbox operational mode" on page 191.

5.3.1 Mailbox command format for a write

Commands written to the MMA955xL platform are sent in the format shown in the following table. Mailboxes are filled with data, depending on the target application. All commands start with the APP_ID, the command, the destination offset, and the number of data bytes to write.

All commands must be written in a single, I^{2}C/SPI transaction starting at Mailbox 0, but the response can be read from any subset of the mailbox registers.

For a write-request command, the first four mailboxes must be written with enough bytes to hold the requested number of bytes. The format of the data bytes is specific to the targeted application. (Applications are described in Table 5-3.)

Table 5-1. Mailbox commands formats

Offset	7	6	5	4	3	2	1	0
0x00	Application ID (APP_ID)							
0x01	0	Command			Byte offset (upper 4 bits)			
0x02	Byte offset (lower 8 bits)							
0x03	Requested number of bytes to read/write							
0x04	Write data 0							
0x05	Write data 1							
0x06	Write data 2							
....	Write data n							

The following table gives the details of the different parts of the data packet for a Write command.

Table 5-2. Mailbox command format details

Block	Description
Application ID	Application targeted for the issued command. (See the next table.)
Command	Command to be performed: • 0: Read application fixed bytes (version information) • 1: Read application configuration bytes • 2: Write application configuration bytes • 3: Read application status or output
Byte offset	Sets the offset of the first byte to be accessed, counting from the start of the register space. This enables a subset of the registers to be accessed by setting the start location to something other than zero.
Requested number of bytes	Number of bytes requested to be read or written.
Write data	The data being written.

5.3.2 Application IDs, names, and descriptions

The following table gives the names and IDs of the Freescale applications associated with MMA95590L, MMA95591L, and MMA95593L platforms.

Table 5-3. Application descriptions[(1)]

Application ID	Application Name	Description
0x00	Version	Returns a 12-byte pack with the device identifier number and the version numbers of the ROM, firmware, and hardware. (For more details, see "Version Application" on page 31.)
0x01	Scheduler	Configures the system, applications, and the MMA955xL infrastructure to run at specific sample rates. Additionally, the identifier reads the number of times each task has been executed.
0x02	Reserved	
0x03	GPIO	Configures the GPIO application to map a specific application output bit to specific GPIO pins. (For more details, see Chapter 6, "GPIO Application"".) The GPIO pins are limited to GPIO6 through 9.
0x04	Mailbox	Configures an internal mailbox table to map which output bytes from specific application identifiers will be accessible in the Normal mode and the Legacy mode's Quick-Read registers mailboxes. The application identifier can perform a table reset to reinstall the default values when the MMA955xL resets.
0x05	Reserved	
0x06	Analog Front End	Configures different parameters of the AFE and reads XYZ data from the accelerometer. For further details, see Chapter 8, "Analog Front End Application".
0x07	Portrait/landscape detection	Configures different parameters of the portrait and landscape application and reads and writes data from and to the applications. For further details, see Chapter 9, "Portrait/Landscape Application".
0x08	High-g detection	Configures different parameters of the high_g application. The application identifier reads the output bytes from its output structure. For further details, see Chapter 10, "High-g/Low-g Application".
0x09	Low-g detection	Configures different parameters of the low_g application. The application identifier reads the output bytes from its output structure. For further details, see Chapter 10, "High-g/Low-g Application".
0x0A	Tap detection	Configures different parameters of the tap-detection application. The application identifier reads the output bytes from its output structure.
0x0B	Tilt sensing	Configures parameters of the tilt-detection application and reads the output bytes from its output structure.
0x0C-0x0D	Available for user applications	Application IDs that can be used for user applications.
0x0E	Frame counter	Reads the system frame counter value.
0x0F	Data FIFO	Configures parameters of the Data-FIFO application and reads the output bytes from its output structure and the contents of the FIFO buffer.

1. Shaded rows indicate unused application identifiers available for user applications.

Table 5-3. Application descriptions[(1)] (continued)

Application ID	Application Name	Description
0x10	Event queue	Configures parameters of the Event-queue application and reads the output bytes from its output structure and the contents of the Event-queue buffer.
0x11	Status register	Provides access to the MMA955xL platform's system-status information.
0x12	Sake/Sleep	Configures the power-control modes of the accelerometer. The application has three modes of operation: Run, Doze, and Sleep.
0x013-0x016	Available for user applications	Application IDs that can be used for user applications.
0x17	Reset/suspend/clear	Controls the Reset/Suspend/Clear functions of the MMA955xL platform.
0x18	Mailbox mode config	Configures different operation modes of the mailbox and provides the status value of the mailbox when Stream mode is running.
0x19 – 0x1F	Available for user applications	Application IDs that can be used for user applications.
0x20 – 0xFF	Reserved	Indicates an invalid application index.

1. Shaded rows indicate unused application identifiers available for user applications.

5.3.3 Mailbox command format for a read

Though all commands must be written in a single I^{2}C/SPI transaction starting at Mailbox 0, the response can be read from any subset of the mailbox registers. When the MMA955xL platform is configured to stream data (as in FIFO mode), the read commands must be constructed as multiples of 32 bytes in order to trigger the internal transfer of the next set of data to the mailboxes.

A read-request command requires a write to the first four mailboxes.

The format of the information returned from the MMA955xL platform is shown in the following table. Similar to the command format, the response format follows the specific application's format.

Mailboxes are filled with data depending on the target application.

All responses start with the responding APP_ID, the COCO the ERROR STATUS, the actual data count, and the requested data count.

The format of the remaining data bytes is specific to the responding application.

Table 5-4. Mailbox response formats

Mailbox	7	6	5	4	3	2	1	0
0x00	Application ID (APP_ID)							
0x01	COCO	Error code						
0x02	Actual number of bytes read/written							
0x03	Requested number of bytes to read/write							
0x04	Read data 0							
0x05	Read data 1							
0x06	Read data 2							
....	Read data n							

The following table describes the details of the different parts and fields of a response message.

Table 5-5. Mailbox response format details

Block	Description
Application ID (APP_ID)	The ID of the application that is responding. (See the preceding table.)
COCO	Command complete. This bit must be set to 0b when a command is written and is set to 1b by the MMA955xL platform, when the command has been processed. The other registers do not contain valid results until this bit is set.
Error code	The seven bytes that store the error code of the command. A zero indicates there was no error. (For more information, see Table 5-6 on page 61.)
Actual number of bytes	Actual number of bytes read or written. This block reports back the actual number of bytes that were read or written. It is normally the same as the requested number of bytes, but it will be reduced if the requested number of bytes plus the Byte Offset exceeds the number of bytes in the requested block's data structure.
Requested number of bytes	Number of bytes requested to be read or written.
Read data	The data that was read.

The following table describes the status or error-code results returned in Mailbox 0x01.

Table 5-6. Error-Status codes returned in Mailbox 0x01

Error Code	Name	Description
0x00	MCI _ERROR_NONE	Command completed with no errors.
0x04	MCI _ERROR_PARAM	Incorrect input parameter. Error may be due to an incorrect application ID, an incomplete command, or an incorrect offset.
0x19	MCI _INVALID_COUNT	Returned when the command COUNT is greater than the output structure size.
0x1C	MCI_ERROR_COMMAND	Returned any time that the command interpreter does not recognize a command code.
0x21	MCI_ERROR_INVALID_LENGTH	Returned when the host sends a number of bytes with a wrong payload. MMA955xL checks any mismatches between the amount of bytes received and the actual payload sent by the host.
0x22	MCI_ERROR_FIFO_BUSY	FIFO is busy performing a push operation and it is not possible to execute any other function.
0x23	MCI_ERROR_FIFO_ALLOCATED	Returned when the host tries to reconfigure the FIFO module. The FIFO application configuration can only be written once. In order to re-configure the FIFO, the whole device must be reset. This is because the FIFO application requests RAM and RAM can only be allocated one time.
0x24	MCI_ERROR_FIFO_OVERSIZE	Returned when the host wants to set a FIFO buffer size out of the memory boundaries within the MMA955xL device.

Chapter 6 GPIO Application

6.1 Overview of GPIO application

Application ID	0x03
Default speed	488 Hz
Configuration registers	Start on page 63.
Status registers	None.

The GPIO application assigns a bit from an application’s status register to a specific GPIO pin. The configuration registers contain the application ID and the bit number of the output byte for each GPIO pin.

The GPIO application connects the MMA955xL platform’s physical GPIO pins to status bits in an applications' status registers. The GPIO application can control four physical GPIO pins. Each of the four GPIO pins (GPIO6, 7, 8, and 9) has an associated APP_ID register and an SR_bit register.

The GPIO application also has a general polarity register where the GPIO pins can be set to be active high or active low. The default or start-up condition of the GPIO pins are unassigned.

The GPIO pins that are controlled by the MMA955xL platform are described in the following table.

Table 6-1. GPIO pin names, numbers and functions

Name	Pin number
GPIO6	12
GPIO7	13
GPIO8	15
GPIO9	2

6.2 GPIO configuration registers

The GPIO application’s configurations registers consist of 10 eight-bit registers. This includes two registers for each of the four GPIO pins and two registers for setting the polarity of the GPIO pin. Each GPIO pin is assigned to an APP_ID and as an output bit from the assigned application.

The following table gives the bit descriptions for the GPIO application’s registers. The application’s registers are shown in Table 6-3 through Table 6-11 on page 65.

The bit descriptions are given in Table 6-13 on page 66.

6.2.1 GPIO register tables

Table 6-2. APP_ID GPIO6 register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID GPIO6							
Reset	0xFF							

Table 6-3. SR_bitnum GPIO6 register

Offset	0x01							
Bit	7	6	5	4	3	2	1	0
Field	SR_bitnum GPIO6							
Reset	0x00							

Table 6-4. APP_ID GPIO7 register

Offset	0x02							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID GPIO7							
Reset	0xFF							

Table 6-5. SR_bitnum GPIO7 register

Offset	0x03							
Bit	7	6	5	4	3	2	1	0
Field	SR_bitnum GPIO7							
Reset	0x00							

Table 6-6. APP_ID GPIO8 register

Offset	0x04							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID GPIO8							
Reset	0xFF							

Table 6-7. SR_bitnum GPIO8 register

Offset	0x05							
Bit	7	6	5	4	3	2	1	0
Field	SR_bitnum GPIO8							
Reset	0x00							

Table 6-8. APP_ID GPIO9 register

Offset	0x06							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID GPIO9							
Reset	0xFF							

Table 6-9. SR_bitnum GPIO9 register

Offset	0x07							
Bit	7	6	5	4	3	2	1	0
Field	SR_bitnum GPIO9							
Reset	0x00							

Table 6-10. GPIO_pol MSB register

Offset	0x08(MSB)							
Bit	7	6	5	4	3	2	1	0
Field	Reserved						GPIO9	GPIO8
Reset	0x00						0	0

Table 6-11. GPIO_pol LSB register

Offset	0x09(LSB)							
Bit	7	6	5	4	3	2	1	0
Field	GPIO7	GPIO6	Reserved					
Reset	0	0	0x00					

6.2.2 GPIO polarity configuration

Table 6-12. GPIO output, depending on polarity configuration and bit value

Input Bits		Output Bit
GPIO_POLARITY	**SR_bit x**	**VAL GPIO x**
0	0	0
0	1	1
1	0	1
1	1	0

6.2.3 GPIO application bit descriptions

Table 6-13. GPIO application bit descriptions

Field	Description
APP_ID GPIOx [7:0]	The application identifier (APP_ID) of the application assigned to the GPIO pin. A value of 0xFF indicates there is no application assigned. After a reset, no application output bits are mapped to a GPIO. Units: None. Range of valid values: [0:31] and 0xFF.
SR_bitnum GPIOx [7:0]	The bit number of the application status registers being assigned to the GPIO pin. The bit number depends of the amount of output bytes and the position of the bit in the register. For example, if the QUADFLAG bit in tilt_xz_yz_xy_quad register of the Tilt algorithm to be assigned, SR_bitnum = 31 because it corresponds to the seventh bit of the fourth output byte. (For more information, see "Tilt Application" on page 127.) Units: None. Range of valid values: [0:255]
GPIO_pol*x*	Defines the output polarity of the GPIO pin designated by the *n* variable in the field name. This register uses negative logic 0 to configure active high and 1 for active low. Table 6-2 on page 64 indicates which pin of the register corresponds to each GPIO. Units: None. Range of valid values: • 0: Output is active high (output bit = high → high on GPIO pin). • 1: Output is active low (output bit = high → low on GPIO pin).
Reserved	Indicates that the bit is reserved.

Chapter 7 Mailbox Application

7.1 Overview of Mailbox application

Application ID	0x04
Default speed	488 Hz
Configuration registers	Start on page 67.
Status registers	None

The Mailbox (MBOX) application gathers output data from other applications and puts that data into the mailbox registers. This enables users to customize and group up to 28 applications' specific data bytes for quick reads.

Normally, the host may need to read the status and output data from many applications. This requires multiple, serial-slave-port transactions. Using the mailbox application enables the host to read all necessary data in one serial, I^{2}C or SPI transaction.

The MBOX application provides a shortcut for the user to read different pieces of data from different applications by combining the data for reading in one I^{2}C or SPI read transaction.

The MBOX application is different than the hardware mailboxes used by the Communication Interface application. The MBOX application combines selected data bytes from specific applications and loads them into the Communications Interface mailboxes.

In order to properly configure the system Communications Interface, both the MBOX Configuration and Mailbox applications also must be properly configured. The MBOX Configuration application controls how the mailboxes behave and the Mailbox application controls what is placed in the mailboxes.

7.2 Mailbox configuration registers

The Mailbox application's registers are described in the following table. The registers' bit descriptions are given in "MBOX bit descriptions" on page 72.

Each response mailbox (MB4 to MB31) has two associated configuration registers: APP_ID register and Byte_ID register. When the Mailbox application runs, it copies the data at the Byte_ID from the specified APP_ID and puts that value in the associated mailbox.

Users can configure mailboxes 4 through 31. Mailboxes 20 through 31 have a special function in Legacy mode. They are updated automatically in the Legacy/Quick-Read mode.

Table 7-1. MBOX registers

Offset	0x00								0x01							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_4								Byte_ID MBOX_4							
Reset	0x00								0x00							
Offset	0x02								0x03							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_5								Byte_ID MBOX_5							
Reset	0x00								0x00							
Offset	0x04								0x05							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_6								Byte_ID MBOX_6							
Reset	0x00								0x00							
Offset	0x06								0x07							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_7								Byte_ID MBOX_7							
Reset	0x00								0x00							
Offset	0x08								0x09							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_8								Byte_ID MBOX_8							
Reset	0x00								0x00							
Offset	0x0A								0x0B							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_9								Byte_ID MBOX_9							
Reset	0x00								0x00							

Table 7-1. MBOX registers (continued)

Offset	0x0C								0x0D							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_10								Byte_ID MBOX_10							
Reset	0x00								0x00							
Offset	0x0E								0x0F							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_11								Byte_ID MBOX_11							
Reset	0x00								0x00							
Offset	0x10								0x11							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_12								Byte_ID MBOX_12							
Reset	0x00								0x00							
Offset	0x12								0x13							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_13								Byte_ID MBOX_13							
Reset	0x00								0x00							
Offset	0x14								0x15							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_14								Byte_ID MBOX_14							
Reset	0x00								0x00							
Offset	0x16								0x17							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_15								Byte_ID MBOX_15							
Reset	0x00								0x00							
Offset	0x18								0x19							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_16								Byte_ID MBOX_16							
Reset	0x00								0x00							

Table 7-1. MBOX registers (continued)

Offset	0x1A								0x1B							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_17								Byte_ID MBOX_17							
Reset	0x00								0x00							
Offset	0x1C								0x1D							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_18								Byte_ID MBOX_18							
Reset	0x00								0x00							
Offset	0x1E								0x1F							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_19								Byte_ID MBOX_19							
Reset	0x00								0x00							
Offset	0x20								0x21							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_20								Byte_ID MBOX_20							
Reset	0x11 (Defaults to Status Register application)								0x00 (Status application - MSB)							
Offset	0x22								0x23							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_21								Byte_ID MBOX_21							
Reset	0x11 (Defaults to Status Register application)								0x01 (Status application - LSB)							
Offset	0x24								0x25							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_22								Byte_ID MBOX_22							
Reset	0x10 (Defaults to Event Queue application)								0x03 (Event Queue Status)							
Offset	0x26								0x27							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_23								Byte_ID MBOX_23							
Reset	0x0F (Defaults to Data FIFO application)								0x03 (FIFO status)							

Table 7-1. MBOX registers (continued)

Offset	0x28								0x29							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_24								Byte_ID MBOX_24							
Reset	0x06 (Defaults to Analog Front End application)								0x28 (Analog Front End Frame Counter - MSB)							
Offset	0x2A								0x2B							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_25								Byte_ID MBOX_25							
Reset	0x06 (Defaults to Analog Front End application)								0x29 (Analog Front End Frame Counter - LSB)							
Offset	0x2C								0x2D							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_26								Byte_ID MBOX_26							
Reset	0x06 (Defaults to Analog Front End application)								0x00 Analog Front End Stage 0 - X MSB							
Offset	0x2E								0x2F							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_27								Byte_ID MBOX_27							
Reset	0x06 (Defaults to Analog Front End application)								0x01 (Analog Front End Stage 0 - X LSB)							
Offset	0x30								0x31							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_28								Byte_ID MBOX_28							
Reset	0x06 (Defaults to Analog Front End application)								0x02 (Analog Front End Stage 0 - Y MSB)							
Offset	0x32								0x33							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_29								Byte_ID MBOX_29							
Reset	0x04 (Analog Front End Stage 0 - Z MSB)								0x03 (Analog Front End Stage 0 - Y LSB)							
Offset	0x34								0x35							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_30								Byte_ID MBOX_30							
Reset	0x06 (Defaults to Analog Front End application)								0x04 (Analog Front End Stage 0 - Z MSB)							

Table 7-1. MBOX registers (continued)

Offset	0x36								0x37							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	APP_ID MBOX_31								Byte_ID MBOX_31							
Reset	0x06 (Defaults to Analog Front End application)								0x05 (Analog Front End Stage 0 - Z LSB)							

7.2.1 MBOX bit descriptions

Table 7-2. MBOX bit descriptions

Field	Description
APP_ID MBOX_*x*	Specifies the application to provide the output byte, using the application identifier. Units: None. Range of valid values: 1 to 31. Values outside this range or values for a nonexistent application may cause unexpected system behavior.
Byte_ID MBOX_*x*	Indicates the byte number of the application’s status or output registers that are being sent to this mailbox. The most-significant byte for a specific application’s output is identified as Byte 0 and the least-significant byte is Byte *x*. Units: Non-dimensional. Range of valid values: 0 to 255.

7.2.2 Configuring XYZ data

The MBOX application aggregates application data and presents it in the mailboxes.

The AFE application (APP_ID 0x06) provides the XYZ accelerometer data with output registers 0 through 5 containing the FRONTEND_Stage_0 XYZ data. The following example shows what a host would send to the MMA955xL device to create the configuration to set up quick-read mailboxes 4–9 to contain the following, FRONTEND_Stage_0_XYZ data:

- Data X to mailboxes 4 and 5
- Data Y to mailboxes 6 and 7
- Data Z to mailboxes 8 and 9

Example 7-1.

```
MBOX0 = 0x04     /* Host communicating to MBOX Application */
MBOX1 = 0x20     /* CONFIG_W command */
MBOX2 = 0x00     /* Starting at Offset 0 which is the config for MB4 */
MBOX3 = 0x0C     /* Number of bytes to write 12 bytes */
MBOX4 = 0x06     /* APP_ID_MBOX4 0x06 - AFE APP_ID = 0x06 */
MBOX5 = 0x00     /* Byte_ID_MBOX4 Data - X MSB */
MBOX6 = 0x06     /* APP_ID_MBOX5 */
MBOX7 = 0x01     /* Byte_ID_MBOX5 Data - X LSB */
MBOX8 = 0x06     /* APP_ID_MBOX6 */
MBOX9 = 0x02     /* Byte_ID_MBOX6 Data - Y MSB */
MBOX10 = 0x06    /* APP_ID_MBOX7 */
MBOX11 = 0x03    /* Byte_ID_MBOX7 Data - Y LSB */
MBOX12 = 0x06    /* APP_ID_MBOX8 */
MBOX13 = 0x04    /* Byte_ID_MBOX8 Data - Z MSB */
MBOX14 = 0x06    /* APP_ID_MBOX9 */
MBOX15 = 0x05    /* Byte_ID_MBOX9 Data Z - LSB */
```

7.3 Mailbox status registers

There are no status registers for this application.

7.4 Reading aggregated data (Legacy mode - Quick read)

Once the MBOX application is configured and the device is set to Legacy communication mode, the aggregated data can be read.

Assuming that the MBOX application was set up as show in Section 7.2.2, “Configuring XYZ data”, the X, Y, and Z acceleration data output will be available in mailboxes 4 through 9, through the normal command/response Communications Interface.

If the host needs to just read the data without the finer control of the command/response model, the MMA955xL can be put into Legacy mode. This assigns the desired data to registers in the Quick-Read section of the Mailbox registers (MB20-MB31).

In Legacy mode, the lower mailbox registers continue to operate in the command/response mode and the upper registers operate in the Quick-Read mode. The data in the Quick-Read registers is automatically updated, so a read-request command is not required before reading the data form the upper mailboxes.

The following examples show how to wake up the device, configure it for quick-reading the low-passed-filtered XYZ data, enable the Legacy mode, and read the data.

The MMA955xL platform comes out of reset in the Low-Power or Sleep mode. In order to start the AFE application and start collecting samples, the MMA955xL must be brought out of Sleep mode and into Run mode

This example shows how to disable Sleep mode and enable Wake mode.

Example 7-2.

```
MBOX0 = 0x12 /* Host communicating to Sleep/Wake Application */
MBOX1 = 0x20 /* CONFIG_Write command */
MBOX2 = 0x06 /* Starting at Offset 0x6 */
MBOX3 = 0x01 /* Number of bytes to write 1 byte */
MBOX4 = 0x00 /* Write 0x00 which wakes up the device */
```

Bytes to Send: 0x12, 0x20, 0x06, 0x01, 0x00

The AFE application (APP_ID 0x06) provides the XYZ accelerometer data with output registers 0 through 5 containing the FRONTEND_Stage_0 XYZ data. By default, the Quick-Read registers (MB26-MB31) are assigned 0x00 Analog Front End Stage 0 - X MSB.

The AFE application, however, provides XYZ, low-pass-filtered data in registers 0x18 through 0x1D (FRONTEND_488_100_LPF). To quickly read this data, the Quick-Read mailbox registers would have to be configured so that they are populated with the low-passed-filtered, XYZ data.

The following example shows how a host would direct the MMA955xL device to set up quick-read mailboxes to contain the following, FRONTEND_488_100_LPG XYZ data:

- Data X to mailboxes 26 and 27
- Data Y to mailboxes 28 and 29
- Data Z to mailboxes 30 and 31

Example 7-3.

```
MBOX0 = 0x04 /* Host communicating to MBOX Application */
MBOX1 = 0x20 /* CONFIG_W command */
MBOX2 = 0x2C /* Starting at Offset 0x2C, the configuration starting point for MB26 */
MBOX3 = 0x0C /* Number of bytes to write 12 bytes */
MBOX4 = 0x06 /* APP_ID_MBOX26 = 0x06 - AFE APP_ID = 0x06 */
MBOX5 = 0x18 /* Byte_ID_MBOX26 = 0x18 - LPF Data starts at register 0x18 - X MSB */
MBOX6 = 0x06 /* APP_ID_MBOX27 */
MBOX7 = 0x19 /* Byte_ID_MBOX27 Data - X LSB */
MBOX8 = 0x06 /* APP_ID_MBOX28 */
MBOX9 = 0x1A /* Byte_ID_MBOX28 Data - Y MSB */
MBOX10 = 0x06 /* APP_ID_MBOX29 */
MBOX11 = 0x1B /* Byte_ID_MBOX29 Data - Y LSB */
MBOX12 = 0x06 /* APP_ID_MBOX30 */
MBOX13 = 0x1C /* Byte_ID_MBOX30 Data - Z MSB */
MBOX14 = 0x06 /* APP_ID_MBOX31 */
MBOX15 = 0x1D /* Byte_ID_MBOX31 Data Z - LSB */
```

Bytes to send: 0x04, 0x20, 0x2C, 0x0C, 0x06, 0x18, 0x06, 0x19, 0x06, 0x1A, 0x06, 0x1B, 0x06, 0x1C, 0x06, 0x1D.

The host then must configure the Mailbox application to operate in Legacy mode, as shown in the following example.

Example 7-4.

1. MB0 = 0x18.

   Sets the “APP_ID: Mailbox Mode Config” application identifier (0x18).
2. MB1 = 0x20.

   Sets the “Command: Write Config” (0x20).
3. MB2 = 0x00.

   Sets the Offset to Zero field (0x00) to point to the configuration register.
4. MB3 = 0x01.

   Sets the Count field to 0x01 because only one data byte needs to be sent.
5. MB4 = 0x10.

   Sets the DATA value to 0x10, which sets the LEGACY field to 1b or Legacy mode.

Bytes to send: 0x18, 0x20, 0x00, 0x01, 0x10.

The MMA955xL platform now is set to Legacy mode and the Quick-Read registers are being constantly updated with the low-passe-filtered, AFE data.

All that remains is the issuing of a command to read the six bytes starting at MB26, which contains the XYZ data in constant-read mode.

In the following example, “0x__” represents the data that is sent back to the host.

Example 7-5.

```
MBOX0 = 0x04 /* Host communicating to MBOX Application */
MBOX1 = 0x30 /* Read Output Data command */
MBOX2 = 0x1A /* Starting at Offset 0x1A, the hexadecimal offset for mailbox 26 */
MBOX3 = 0x06 /* Number of bytes to read 6 bytes two bytes each for X, Y, andZ*/
MBOX4 = 0x__ /* MSB  - X */
MBOX5 = 0x__ /* LSB - X */
MBOX6 = 0x__ /* MSB - Y */
MBOX7 = 0x__ /* LSB - Y */
MBOX8 = 0x__ /* MSB - Z */
MBOX9 = 0x__ /* LSB - Z*/
```

Bytes to send: 0x04, 0x30, 0x1A, 0x06, 0x__, 0x__, 0x__, 0x__, 0x__, 0x__.

Chapter 8 Analog Front End Application

8.1 Overview of Analog Front End application

Application ID	0x06
Default speed	488 Hz
Configuration registers	Start on page 83.
Status registers	Start on page 87.

The Analog Front End application (AFE) samples raw accelerometer data from the analog-to-digital converter (ADC) at the execution rate of the application, applies factory and user trim correction terms, and filters data to several configurable bandwidths.

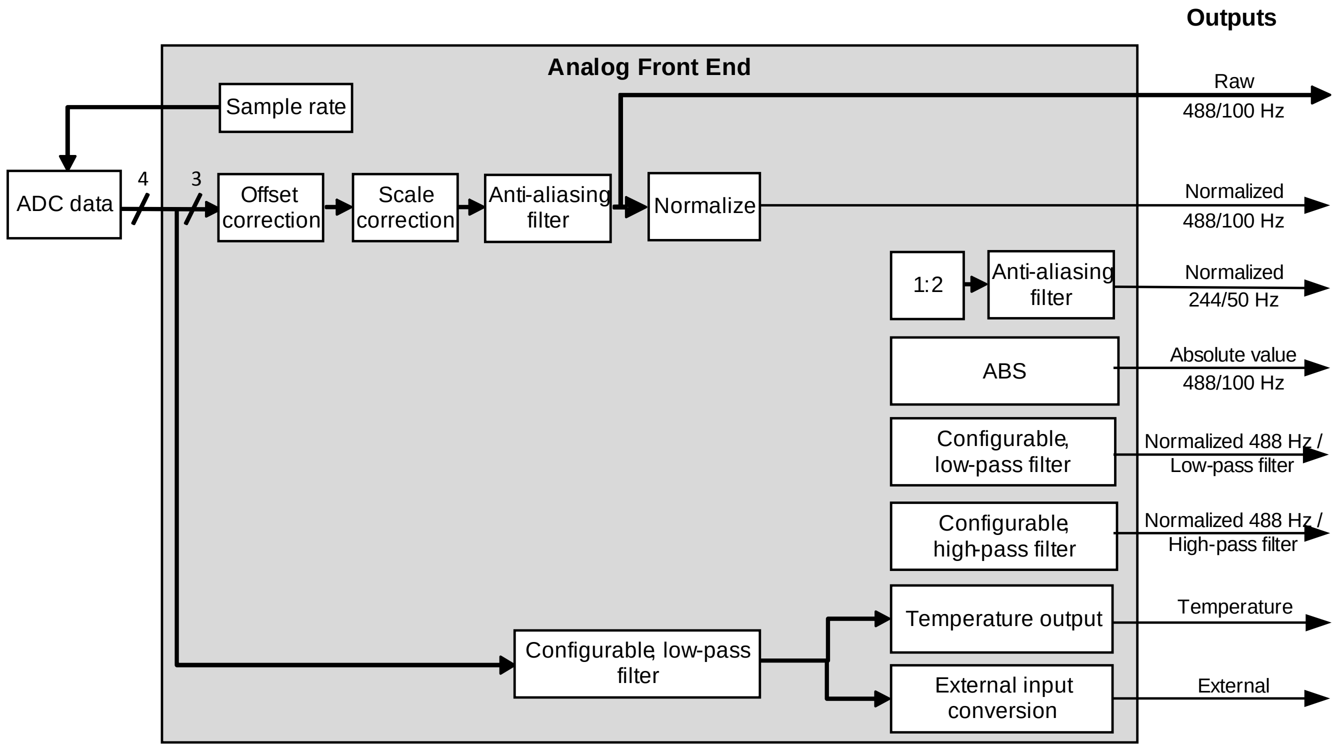


Figure 8-1. Front-end signal processing

8.1.1 Sample rate

The rate at which the ADC samples the accelerometer data is defined by the sfd_rate register, which is offset 0x0C in the configuration registers of this application. The sample rate can be changed from 488 Hz to 3.81 Hz by setting the appropriate value in the sfd_rate register. For more details, see Example 8-12 on page 86 and Table 8-13 on page 86.

NOTE

Although the sample rate can be changed in the hardware, it is recommended that the 488-Hz sample rate not be changed because the system’s other applications expect the sample rate to be 488 Hz.

8.1.2 Offset and scale correction

The AFE’s offset and scale correction stages of the signal chain applies trim offset and scaling correction factors which were measured at factory calibration and stored inside each device. Since user offsets are defined at 8-g resolution, those offsets are shifted according to the g mode.

User offsets are used to calibrate and compensate for the physical mounting of the part inside the final product. After board mount and assembly, the final test process may include a test to fine tune and compensate the accelerometer orientation.

8.1.3 Anti-aliasing filter

After trim correction, the front end uses a sixth-order Chebyshev filter, running at 488 Hz, to limit the signal bandwidth to 100 Hz.

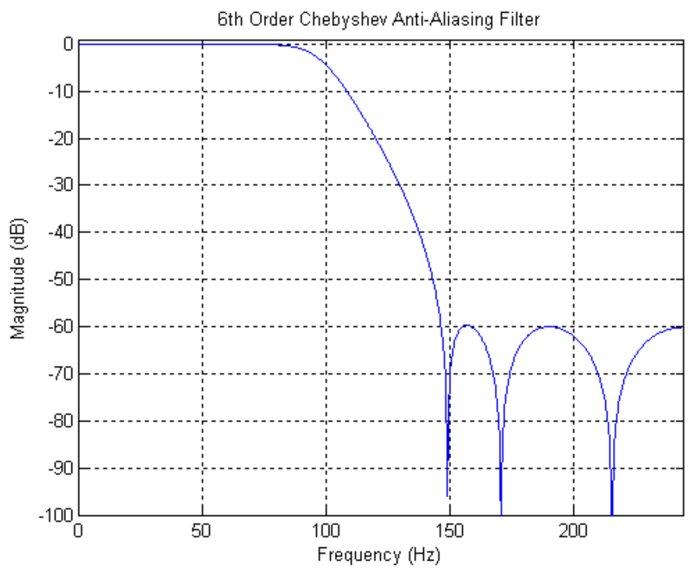


Figure 8-2. Frequency response of Analog Front End anti-aliasing filter

The bandwidth of the anti-aliasing filter depends on the sample rate at which the front end application is running. The following table shows the variation of the filter bandwidth according to the sample rate.

Table 8-1. Anti-aliasing filter bandwidth for different sample rates

Sample rate	Stage-0 anti-aliasing Filter	Stage-1 anti-aliasing Filter
	Bandwidth (Hz)	Bandwidth (Hz)
488.28	100	50
244.17	50	25
122.07	25	12.5
61.04	12.5	6.25
31.52	6.25	3.125
15.26	3.125	1.562
7.63	1.562	0.781
3.81	0.781	0.390

8.1.4 Raw data

The output of the sensor is a 16-bit, signed value. The sensor always uses all the bits for high accuracy. Depending on the range setting—either 2, 4, or 8 g mode—the output value per g changes.

The following table shows the full scale value at the different g ranges, as well as the value of a measured 1g acceleration at the different g ranges.

Table 8-2. Raw accelerometer output, according to g mode

Range	Full scale	1-g acceleration
+/- 2g	± 32K	16K
+/- 4g	± 32K	8K
+/- 8g	± 32K	4K

8.1.5 Normalization

The filtered data is shifted according to the g mode to normalize the resolution to the 8-g range. The normalized data allows for common handling of the data in the other applications. Data is normalized so that 1-g force acceleration shows output of 0x1000 counts or 4096 decimal counts.

8.1.6 Down-sampling and stage-1, anti-aliasing filter

The output of the normalizer is down-sampled by two to generate data sampled at 244 Hz. The previous, sixth-order Chebyshev filter is applied to down-sampled (244 Hz) data, to create a 50-Hz bandwidth data stream.

Table 8-13 provides more details about the varying of the bandwidth, depending of the sample rate of the front-end application.

8.1.7 Absolute value

The absolute value of the sensor output is computed.

8.1.8 Configurable, low-pass and high-pass filters

First-order, low-pass and high-pass filters—with separate configurable cutoff frequencies at -3 dB—are provided.

$$H_{LPF}(z) = \frac{2^{-K}}{1 + (2^{-K} - 1)z^{-1}} \qquad \text{Eqn. 8-1}$$

$$H_{HPF}(z) = \frac{1 - z^{-1}}{1 + (2^{-K} - 1)z^{-1}} \qquad \text{Eqn. 8-2}$$

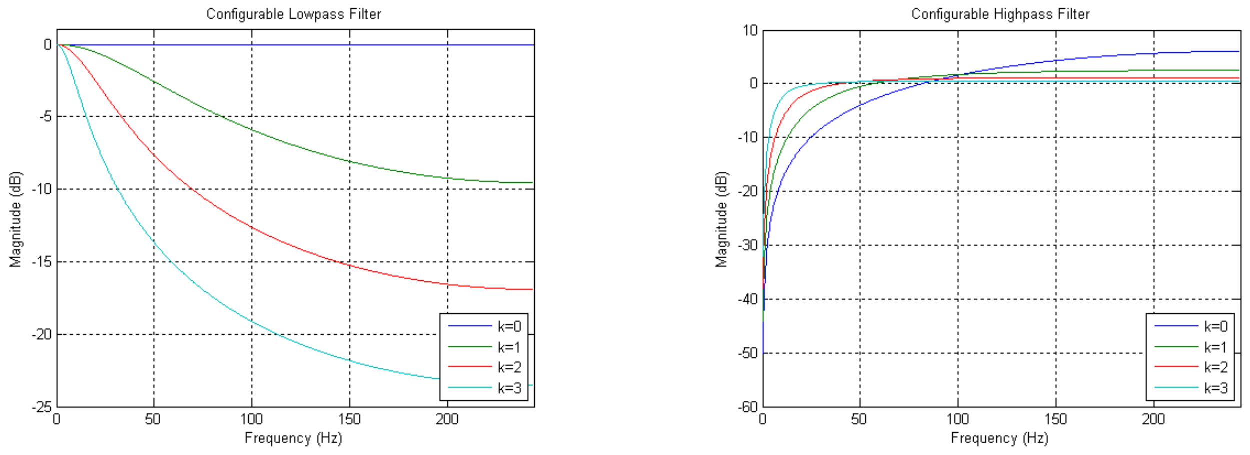


Figure 8-3. Frequency response of configurable, high- and low-pass filters

The two following tables contain the cut-off frequency of the filters, varying with the value of K and the sample rate at which the front-end application is executed.

Table 8-3. Cut-off frequency as a function of K and the sample rate for low-pass filter

K	LPF cut-off frequency (Hz)							
	Sample rate = 488.28	**Sample rate = 244.14**	**Sample rate = 122.07**	**Sample rate = 61.04**	**Sample rate = 30.52**	**Sample rate = 15.26**	**Sample rate = 7.63**	**Sample rate = 3.81**
1	56.13	28.07	14.03	7.02	3.51	1.75	0.88	0.4385
2	22.50	11.25	5.62	2.81	1.40	0.70	0.3516	0.1758
3	10.39	5.19	2.60	1.30	0.65	0.32	0.1623	0.0811
4	5.014	2.51	1.25	0.63	0.31	0.16	0.0783	0.0392
5	2.47	1.23	0.62	0.31	0.15	0.08	0.0385	0.0193
6	1.22	0.6116	0.31	0.15	0.08	0.04	0.0191	0.0096
7	0.61	0.30	0.15	0.08	0.04	0.02	0.0095	0.0048
8	0.30	0.15	0.08	0.04	0.02	0.01	0.0048	0.0024
9	0.15	0.076	0.04	0.02	0.01	0.0047	0.0024	0.0012
10	0.0759	0.038	0.019	0.0095	0.0047	0.0024	0.0012	0.0006
11	0.0379	0.019	0.0095	0.0047	0.0024	0.0012	0.0006	0.0003
12	0.019	0.0095	0.0048	0.0024	0.0012	0.0006	0.0003	0.0001
13	0.0095	0.0048	0.0024	0.0012	0.0006	0.0003	0.0001	0.00007
14	0.0047	0.0024	0.0012	0.0006	0.0003	0.0001	0.0007	0.00004
15	0.0024	0.0012	0.0006	0.0003	0.0002	0.00008	0.00004	0.00002

Table 8-4. Cut-off frequency as a function of K and the sample rate for high-pass filter

K	HPF cut-off frequency (Hz)							
	Sample rate = 488.28	**Sample rate = 244.14**	**Sample rate = 122.07**	**Sample rate = 61.04**	**Sample rate = 30.52**	**Sample rate = 15.26**	**Sample rate = 7.63**	**Sample rate = 3.81**
1	31.932	15.966	7.9831	3.9915	1.9958	0.9979	0.4989	0.2495
2	17.405	8.7023	4.3511	2.1756	1.0878	0.5439	0.2719	0.136
3	9.1573	4.5787	2.2893	1.1447	0.5723	0.2862	0.1431	0.0715
4	4.7092	2.3546	1.1773	0.5887	0.2943	0.1472	0.0736	0.0368
5	2.3912	1.1956	0.5978	0.2989	0.1495	0.0747	0.0374	0.0187
6	1.2054	0.6027	0.3014	0.1507	0.0753	0.0377	0.0188	0.0094
7	0.6051	0.3026	0.1513	0.0756	0.0378	0.0189	0.0095	0.0047
8	0.3026	0.1513	0.0757	0.0378	0.0189	0.0095	0.0047	0.0024
9	0.1513	0.0757	0.0378	0.0189	0.0095	0.0047	0.0024	0.0012
10	0.0756	0.0378	0.0189	0.0095	0.0047	0.0024	0.0012	0.0006
11	0.039	0.0195	0.0098	0.0049	0.0024	0.0012	0.0006	0.0003
12	0.0195	0.0098	0.0049	0.0024	0.0012	0.0006	0.0003	0.0002
13	0.0098	0.0049	0.0025	0.0012	0.0006	0.0003	0.0002	8E-05
14	0.0049	0.0025	0.0012	0.0006	0.0003	0.0002	8E-05	4E-05
15	0.0024	0.0012	0.0006	0.0003	0.0002	8E-05	4E-05	2E-05

8.2 AFE configuration registers

8.2.1 afe_csr

Table 8-5. afe_csr registers

Offset	0x00(MSB)								0x01(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	FS		Ext ADC	Tem p	CM		Reserved		Reserved							
Reset	0x00		0	0	0x00		0x00		0x00							

Table 8-6. afe_csr bit descriptions

Field	Description
FS	Full-scale selection. AFE_CSR[FS] and AFE_BIAS[SC_AAF_EN] are combined to control AFE gain and tgrim mode. Units: Note. Range of values: For valid values, see the MMA955xL *3-Axis Accelerometer Reference Manual* (MMA955xLRM).
Ext ADC	Specifies whether an external input will be measured during the next analog acquisition phase. **Note:** The ExtADC and Temp bits must not be set at the same time. Units: None. • 0: No external input is measured during the next analog acquisition phase. • 1: Enables the four AFE channels to measure the external analog input during the next analog acquisition phase.
Temp	Specifies whether the temperature sensor output will be measured during the next analog acquisition phase. **Note:** The ExtADC and Temp bits must not be set at the same time. Units: None. Temperature will change slowly so it can be measured with a very low sample rate (1Hz or less) by occasionally replacing an external ADC measurement with the Temperature ADC measurement. This could be done with a customized user application. Range of valid values: For valid values for FS and CM parameters, see the MMA955xL *3-Axis Accelerometer Reference Manual* (MMA955xLRM). • 0: No temperature sensor output will be measured during the next analog acquisition phase. • 1: The temperature sensor output will be measured during the next analog acquisition phase.
CM	Conversion Mode. Controls the ADC resolution/accuracy versus power and conversion time trade-offs. Units: None • 00: Conversion complete in 32 cycles • 01: Conversion complete in 16 cycle • 10: Conversion complete in eight cycles • 11: Conversion complete in four cycles
Reserved	Bit field reserved.

8.2.2 user_offset [XYZ]

Often, due to the user manufacturing process, the accelerometer sensor is not mounted perfectly flat to the board and may also be rotated slightly. This register enables a user to make an after-manufacturing calibration correction.

Table 8-7. user_offset [XYZ] registers

Offset	0x02(MSB)								0x03(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	user_offset[X]								user_offset[X][							
Reset	0xFF								0xFF							
Offset	0x04(MSB)								0x05(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	user_offset[Y]								user_offset[Y]							
Reset	0xFF								0xFF							
Offset	0x06(MSB)								0x07(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	user_offset[Z]								user_offset[Z]							
Reset	0xFF								0xFF							

Table 8-8. user_offset [XYZ] bit descriptions

Field	Description
user_offset[XYZ]	Sets user offsets in the X, Y, and Z axes for the accelerometer, depending on the position on the device in the user board. These values are considered as post-mount offsets. Units: 0.244 mg/LSB. Range of valid values: -32,768 to 32,767. Reset value: 0xFF which is interpreted as -1.

8.2.3 config_k

This register's bits are shown in the following table.

Table 8-9. config_k registers

Offset	0x08								0x09							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	config_k[HIGHPASS]								config_k[LOWPASS]							
Reset	0x00								0x00							
Offset	0x0A								0x0B							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	config_k[TEMP_LPF]								config_k[EIC_LPF]							
Reset	0x00								0x00							

Table 8-10. config_k bit descriptions

Field	**Description**
config_k [HIGHPASS]	High-pass filter configurable cutoff. Units: None. Range of valid values: 0 to 15.
config_k [LOWPASS]	Low-pass filter configurable cutoff. Units: None. Range of valid values: 0 to 15.
config_k [TEMP_LPF]	Low-pass filter, configurable cutoff for temperature sensor output. Units: None. Range of valid values: 0 to 15.
config_k [EIC_LPF]	Low-pass filter, configurable cutoff for external input conversion output. Units: None. Range of valid values: 0 to 15.

8.2.4 sfd_rate

This register’s bits are shown in the following table. The system defaults to 488 Hz, so this document assumes that is the rate being run. If the user slows down the rate by changing the sfd_rate register, all references to 488 Hz or the system frame rate will scale accordingly.

Table 8-11. sfd_rate register

Offset	0x0C							
Bit	7	6	5	4	3	2	1	0
Field	sfd_rate							
Reset	0	0	0	0	0	1	1	1

Table 8-12. sfd_rate bit description

Field	Description
sfd_rate	Defines the system frame interval, the time period for each sample or the system sample rate in Hz. Units: None. Range of valid values: 7 to 14. Table 8-12 shows the relation between the register value and the interval between each frame.

Table 8-13. Frame interval, according to the sfd_rate value

sfd_rate value	Time frame (s)	Max frames per second
7	2.05E-3	488.28
8	4.10E-3	244.17
9	8.19E-3	122.07
10	1.64E-2	61.04
11	3.28E-2	30.52
12	6.55E-2	15.26
13	1.31E-1	7.63
14	2.62E-1	3.81

8.3 AFE status registers

8.3.1 output[FRONTEND_Stage_0][XYZ]

These registers' bits are shown in the following table.

Table 8-14. output[FRONTEND_Stage_0][XYZ] registers

Offset	0x00(MSB)								0x01(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0][X]								output[FRONTEND_Stage_0][X]							
Reset	0x00								0x00							
Offset	0x02(MSB)								0x03(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0][Y]								output[FRONTEND_Stage_0][Y]							
Reset	0x00								0x00							
Offset	0x04(MSB)								0x05(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0][Z]								output[FRONTEND_Stage_0][Z]							
Reset	0x00								0x00							

Table 8-15. output[FRONTEND_Stage_0][XYZ] bit description

Field	Description
output [FRONTEND_Stage_0] [XYZ]	Normalized accelerometer data sampled at the default rate of 488 Hz, with a 100-Hz bandwidth. Units: 0.244 mg/LSB. Range of valid values: -32,768 to 32,767.

8.3.2 output[FRONTEND_Stage_1][XYZ]

These registers' bits are shown in the following table.

Table 8-16. output[FRONTEND_Stage_1][XYZ] registers

Offset	0x06(MSB)								0x07(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_1][X]								output[FRONTEND_Stage_1][X]							
Reset	0x00								0x00							
Offset	0x08(MSB)								0x09(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_1][Y]								output[FRONTEND_Stage_1][Y]							
Reset	0x00								0x00							
Offset	0x0A(MSB)								0x0B(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_1][Z]								output[FRONTEND_Stage_1][Z]							
Reset	0x00								0x00							

Table 8-17. output[FRONTEND_Stage_1][XYZ] bit description

Field	Description
output [FRONTEND_Stage_1] [XYZ]	Normalized accelerometer data sampled at the default rate of 244 Hz, with a 50-Hz bandwidth. Units: 0.244 mg/LSB. Range of valid values: -32,768 to 32,767.

8.3.3 output[FRONTEND_Stage_0_ABS][XYZ]

These registers' bits are shown in the following table.

Table 8-18. output[FRONTEND_Stage_0_ABS][XYZ] registers

Offset	0x0C(MSB)								0x0D(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_ABS][X]								output[FRONTEND_Stage_0_ABS][X]							
Reset	0x00								0x00							
Offset	0x0E(MSB)								0x0F(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_ABS][Y]								output[FRONTEND_Stage_0_ABS][Y]							
Reset	0x00								0x00							
Offset	0x10(MSB)								0x11(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_ABS][Z]								output[FRONTEND_Stage_0_ABS][Z]							
Reset	0x00								0x00							

Table 8-19. output[FRONTEND_Stage_0_ABS][XYZ] bit description

Field	Description
output [FRONTEND_Stage_0_ABS] [XYZ]	Absolute value normalized accelerometer data sampled at the default rate of 488 Hz, with a 100-Hz bandwidth. Units: 0.244 mg/LSB. Range of valid values: 0 to 32,767.

8.3.4 output[FRONTEND_Stage_0_GM][XYZ]

These registers' bits are shown in the following table.

Table 8-20. output[FRONTEND_Stage_0_GM][XYZ] registers

Offset	0x12(MSB)								0x13(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_GM][X]								output[FRONTEND_Stage_0_GM][X]							
Reset	0x00								0x00							
Offset	0x14(MSB)								0x15(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_GM][Y]								output[FRONTEND_Stage_0_GM][Y]							
Reset	0x00								0x00							
Offset	0x16(MSB)								0x17(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_GM][Z]								output[FRONTEND_Stage_0_GM][Z]							
Reset	0x00								0x00							

Table 8-21. output[FRONTEND_Stage_0_GM][XYZ] bit description

Field	Description
output [FRONTEND_Stage_0_GM] [XYZ]	Raw accelerometer data sampled at the default rate of 488 Hz, with a 100-Hz bandwidth. The resolution depends on the g-mode setting configured by afe_csr[fs]. Units: • 2g mode: 0.061 mg/LSB • 4g mode: -.122 mg/LSB • 8g mode: 0.244 mg/LSB Range of valid values: • +/- 2g Mode: 0.061mg/LSB = -32768 to 32767 • +/- 4g Mode: 0.122mg/LSB = - 8192 to 8192 • +/- 8g Mode: 0.244mg/LSB = -4096 to 4096

8.3.5 output[FRONTEND_Stage_0_LPF][XYZ]

These registers' bits are shown in the following table.

Table 8-22. output[FRONTEND_Stage_0_LPF][XYZ] registers

Offset	0x18(MSB)								0x19(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_LPF][X]								output[FRONTEND_Stage_0_LPF][X]							
Reset	0x00								0x00							
Offset	0x1A(MSB)								0x1B(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_LPF][Y]								output[FRONTEND_Stage_0_LPF][Y]							
Reset	0x00								0x00							
Offset	0x1C(MSB)								0x1D(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_LPF][Z]								output[FRONTEND_Stage_0_LPF][Z]							
Reset	0x00								0x00							

Table 8-23. output[FRONTEND_Stage_0_LPF][XYZ] bit description

Field	Description
output [FRONTEND_Stage_0_LPF] [XYZ]	Normalized accelerometer data sampled at the default rate of 488 Hz mode, with configurable, low-pass filter cutoff. Units: 0.244 mg/LSB. Range of valid values: -32,768 to 32,767.

8.3.6 output[FRONTEND_Stage_0_HPF][XYZ]

These registers' bits are shown in the following table.

Table 8-24. output[FRONTEND_Stage_0_HPF][XYZ] registers

Offset	0x1E(MSB)								0x1F(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_HPF][X]								output[FRONTEND_Stage_0_HPF][X]							
Reset	0x00								0x00							
Offset	0x20(MSB)								0x21LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_HPF][Y]								output[FRONTEND_Stage_0_HPF][Y]							
Reset	0x00								0x00							
Offset	0x22(MSB)								0x23(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output[FRONTEND_Stage_0_HPF][Z]								output[FRONTEND_Stage_0_HPF][Z]							
Reset	0x00								0x00							

Table 8-25. output[FRONTEND_Stage_0_HPF][XYZ] bit description

Field	Description
output [FRONTEND_Stage_0_HPF] [XYZ]	Normalized accelerometer data sampled at the default rate of 488 Hz mode, with configurable, high-pass filter cutoff. Units: 0.244 mg/LSB. Range of valid values: -32,768 to 32,767.

8.3.7 output_temp

This register's bits are shown in the following table.

Table 8-26. output_temp registers

Offset	0x24(MSB)								0x25(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output_temp								output_temp							
Reset	0x00								0x00							

Table 8-27. output_temp bit description

Field	Description
output_temp	Temperature sensor output measurement. Units: ADC counts. Range of valid values: -32,768 to 32,767.

8.3.8 output_EIC

This register's bits are shown in the following table.

Table 8-28. output_EIC registers

Offset	0x26(MSB)								0x27(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	output_EIC								output_EIC							
Reset	0x00								0x00							

Table 8-29. output_EIC bit description

Field	Description
output_EIC	Output of the external, analog-input conversion value. Units: 3.472 mV/LSB. Range of valid values: -32,768 to 32,767.

8.3.9 frame_counter

This register’s bits are shown in the following table.

The frame_counter register also can be read with the Frame Counter application. See Chapter 13, “Frame Counter Application” on page 135.

Table 8-30. frame_counter registers

Offset	0x28(MSB)								0x29(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	frame_counter								frame_counter							
Reset	0x00								0x00							

Table 8-31. output_EIC bit description

Field	Description
frame_counter	Provides the number of frames that have been processed at the configured sample rate. It is not a real-time representative because the time that the device remains in the stop mode is not taken into account. The frame counter will restart at zero when it reaches 65,535. Units: Non-dimensional. Range of valid values: 0 to 65,535.

Chapter 9 Portrait/Landscape Application

9.1 Overview of Portrait/Landscape application

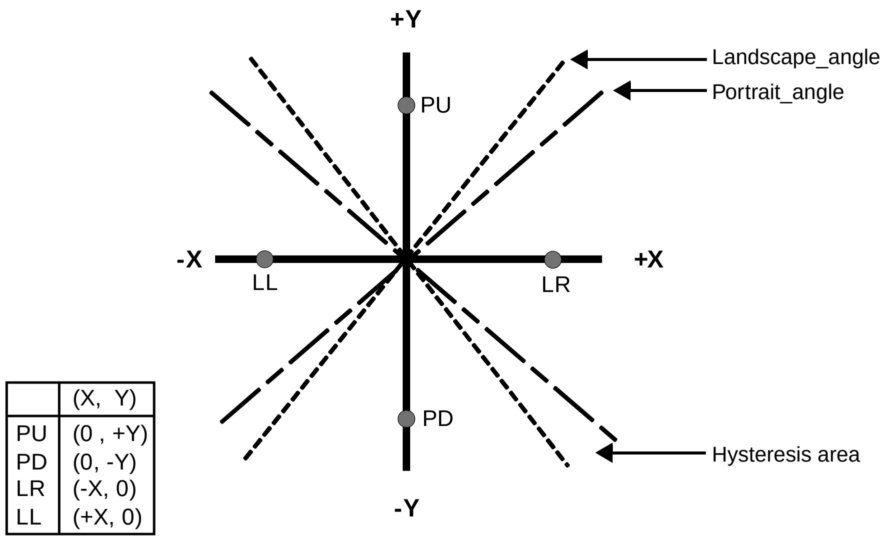
Application ID	0x07
Default speed	122 Hz
Configuration registers	Start on page 98.
Status registers	Start on page 102.

The Portrait/Landscape (PL) application detects positions in two and three dimensions. In the two-dimensional X/Y plane, it determines whether the device orientation is portrait up or portrait down and landscape right or landscape left (PU, PD, LR, and LL, respectively). In the third, z-axis dimension, the application detects whether the device is facing front or facing back.

The PL application uses outputs from the Tilt application (APP_ID = 0x0B) to determine the position of the device.

This application assumes a specific mounting of the MMA955xL platform on a board. The meanings of up, down, front, and back can change with the device's mounting on the board.

The following figure shows PL orientations in the X/Y plane:.



	(X, Y)
PU	(0 , +Y)
PD	(0, -Y)
LR	(-X, 0)
LL	(+X, 0)

Figure 9-1. Portrait and landscape detection in the X/Y plane

The preceding figure shows the axes at which the landscape and portrait positions are detected and the polarities of those positions. The angles of the axes are configurable.

The region between the landscape and portrait axes is the hysteresis area where the previous state remains reported until the second axis is crossed.

The PL application determines the position by comparing the X/Y angle, obtained from the tilt application, with landscape and portrait configured angles. The application uses the quadrant value to select direction up or down, for portrait, or right or left, for landscape.

The following figure shows the package orientation and measured values reported by the tilt application. In the Portrait Up orientation, both X and Z axes read 0g and the Y axis reads back the equivalent of -1g.

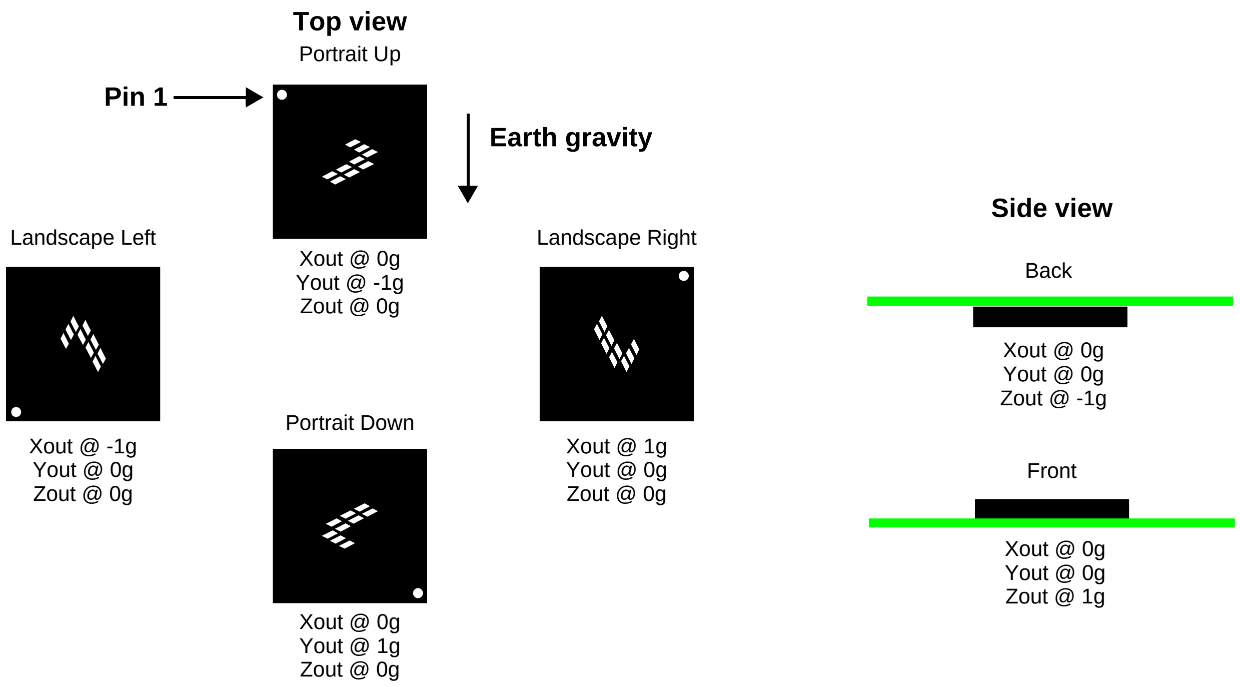


Figure 9-2. Sensing direction and output response

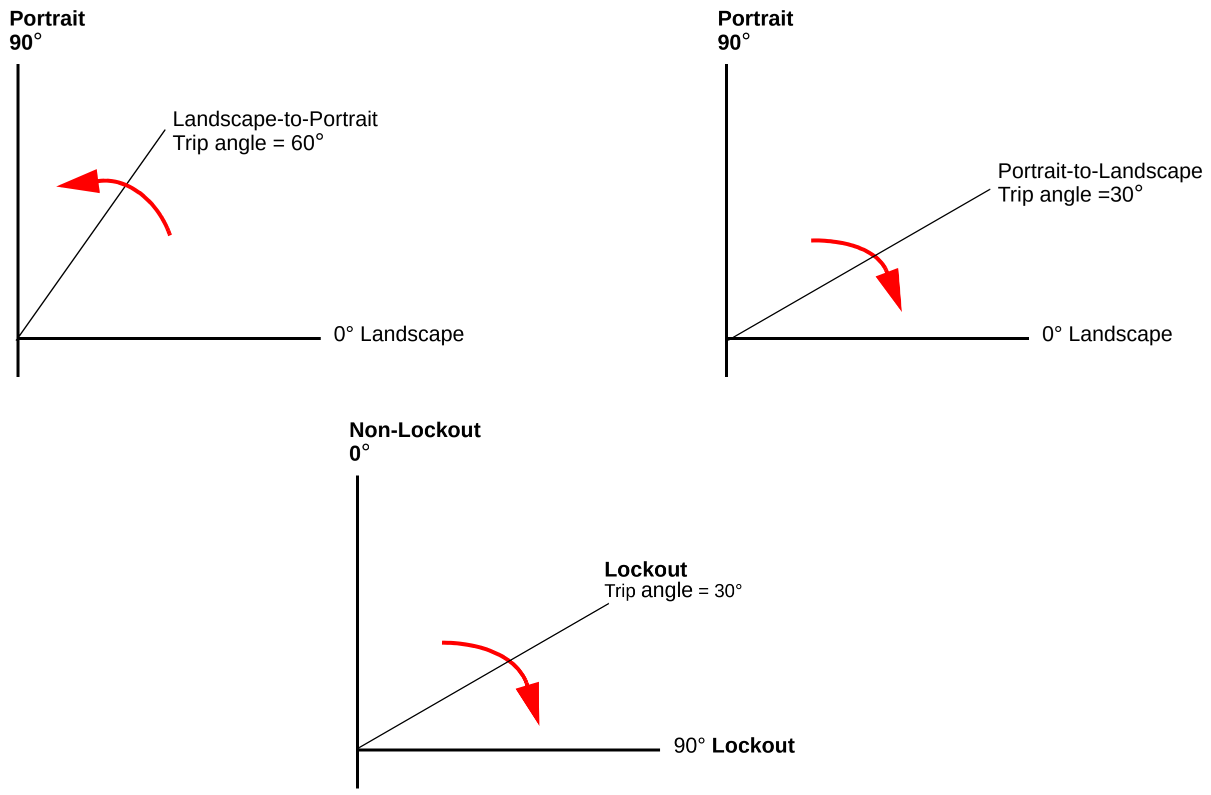


Figure 9-3. Angles for portrait and landscape detection

The 3D detection for back or front is done using the sign of the z axis. The PL application returns positive z for the back orientation and a negative z for front orientation.

9.2 Portrait/Landscape configuration registers

9.2.1 threshold_tilt

Table 9-1. threshold_tilt register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	threshold_tilt							
Reset	0	0	1	0	0	0	1	1

Table 9-2. threshold_tilt bit description

Field	Description
threshold_tilt	Determines the angle of the Z axis at which the application enters and exits the lock-out state. Units: Degrees. Range of valid values: 0 to 90.

9.2.2 landscape_angle

Table 9-3. landscape_angle register

Offset	0x01							
Bit	7	6	5	4	3	2	1	0
Field	landscape_angle							
Reset	0	0	1	0	1	1	0	0

Table 9-4. landscape_angle Bit Description

Field	Description
landscape_angle	Determines the angle in the X/Y plane at which the application changes between portrait and landscape. Units: Degrees. Range of valid values: 0 to 90.

9.2.3 portrait_angle

Table 9-5. portrait_angle register

Offset	0x02							
Bit	7	6	5	4	3	2	1	0
Field	portrait_angle							
Reset	0	0	1	0	1	1	0	1

Table 9-6. portrait_angle bit description

Field	Description
portrait_angle	Determines the angle in the X/Y plane at which the application changes between landscape and portrait. Units: Degrees. Range of valid values: 0 to 90.

9.2.4 debounce_count

Table 9-7. debounce_count register

Offset	0x03							
Bit	7	6	5	4	3	2	1	0
Field	debounce_count							
Reset	0	0	1	0	0	0	0	0

Table 9-8. debounce_count bit description

Field	Description
debounce_count	Sets the number of continuous counts required in a state before changing to the new state. This avoids false decisions of position for high frequencies. Units: Samples. Range of valid values: 0 to 255.

9.2.5 hysteresis_LO

Table 9-9. hysteresis_LO register

Offset	0x04							
Bit	7	6	5	4	3	2	1	0
Field	hysteresis_LO							
Reset	0x00							

Table 9-10. hysteresis_LO bit description

Field	Description
hysteresis_LO	Configures the hysteresis for the lockout state. Units: Degrees. Range of valid values: 5 to 40.

9.2.6 hysteresis_BAFRO

Table 9-11. hysteresis_BAFRO register

Offset	0x05							
Bit	7	6	5	4	3	2	1	0
Field	hysteresis_BAFRO							
Reset	1	0	1	0	1	1	1	1

Table 9-12. hysteresis_BAFRO bit description

Field	Description
hysteresis_BAFRO	Configures the hysteresis for back and front angle transition. This value can be calculated, based on the desired angle, using the following formula: $\text{hysteresis\_BAFRO} = \frac{(\sin(\text{desired angle}) \cdot 4096)}{8}$ Where the desired angle is between 0 and 30°. Units: Non-dimensional. Range of valid values: 0 to 255.

9.2.7 cfg

Table 9-13. cfg register

Offset	0x06							
Bit	7	6	5	4	3	2	1	0
Field	PLFDE	PL_EN	BKFR_EN	—	DBCNTM	—		
Reset	0	1	1	0	0	0		

Table 9-14. cfg bit descriptions

Field	Description
PLFDE	Indicates if an event is saved into the Event Q. • 0: No event saved to Event Q. • 1: Event saved to Event Q.
PL_EN	Enables 2D functionality. • 0: 2D not enabled. • 1: 2D enabled
BKFR_EN	Enables 3D functionality. • 0: 3D not enabled. • 1: 3D enabled
—	Reserved.
DBCNTM	Debounce counter mode. • 0: The debounce counter decrements when the condition is no longer valid. • 1: The counter clears when the condition is no longer valid.
—	Reserved.

9.3 Portrait/Landscape status registers

9.3.1 PL_Out

Table 9-15. PL_Out register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	Evnt	Lock	*	LAPO			BAFRO	
Reset	0	0	0	0x00			0x00	

Table 9-16. PL_Out bit descriptions

Field	Description
Evnt	Reports a PL event. • 0: No event detected. • 1: PL change orientation event detected.
Lock	Reports a z-tilt angle lockout • 0: Lockout condition has not been detected. • 1: Z-tilt lockout trip angle has been exceeded. Lockout has been detected .
LAPO	Reports a new portrait/landscape orientation event. • 000: Undefined. The default, power-up state. • 001: Portrait up – Device is standing vertically in the normal orientation. • 010: Portrait down – Device is standing vertically in the inverted orientation. • 011: Landscape right – Device is in the landscape mode to the right. • 100: Landscape left – Device is in the landscape mode to the left.
BAFRO	Reports a new back or front orientation event. • 00: Undefined – This is the default power up state. • 01: Front – Device is in the front-facing orientation. • 10: Back – Device is in the back-facing orientation.

9.3.2 Programming example

The following example shows how the PL application could be connected to a GPIO pin. In the code, the PL application is APP_ID = 0x07 and the bit that indicates an orientation changes is bit 7 of the PL application's status register.

If the user wants to toggle GPIO Pin 8 whenever an orientation change is detected, the example shows what a host should send to the MMA955xL device to create this configuration.

Example 9-1. Connecting the PL application to a GPIO pin

```
MBOX0 = 0x03       /*GPIO Application ID*/
MBOX1 = 0x20       /*CONFIG_W command*/
MBOX2 = 0x04       /*Starting at Offset 4 for the GPIO 8 registers*/
MBOX3 = 0x02       /*Number of bytes to write 12 bytes*/
MBOX4 = 0x07       /*APP_ID_GPIO8 =  0x07 for the PL Application */
MBOX5 = 0x07       /* SR_bitnum_GPIO8 = 0x07 for the 7th bit in the PL Application
                    status which is the PL Event.*/
```

Chapter 10 High-g/Low-g Application

10.1 Overview of High-g/Low-g application

Application ID	0x08 (high) 0x09 (low)
Default speed	244 Hz
Configuration registers	Start on page 108.
Status registers	Start on page 115.

This application reads the acceleration values of each axis, tests the value for each axis, and determines if the value is above the high-g threshold or below the low-g threshold. The application outputs a bit for each axis value detected as low-g or high-g. The application also outputs a bit with the high-g flags ORed or ANDed and one bit for the low-g flags ORed or ANDed.

High-g and low-g thresholds are configurable by the user.

The High-g/Low-g application uses the 3-axis data from the accelerometer and transforms it into absolute values. These values are used to determine if any of the axes values are above the high-g threshold or below the low-g threshold.

For user convenience, this application appears at both APP_IDs 0x08 and 0x09. The High-g and Low-g configurations and outputs are all in the same registers, at both of the APP_IDs.

The following figure shows the high-g/low-g block diagram.

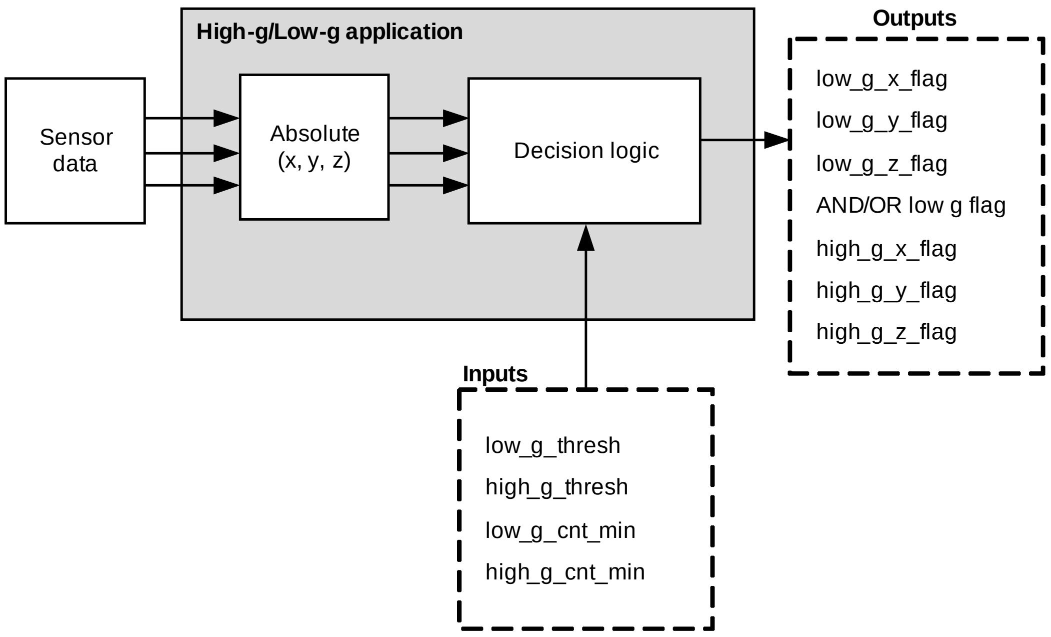


Figure 10-1. High-g and low-g signal-flow diagram

When the signal goes above the high-g threshold for the minimum number of times configured in the high_g_cnt_min register, the application detects a high-g condition for the single axis. When the signal goes below the low-g threshold for the minimum number of times configured in the low_g_cnt_min register, a low-g condition is detected for a single axis.

When the high-g/low-g condition is detected, logic starts incrementing a debounce counter. If the signal is no longer detected as high-g or low-g and the counter did not reach the high_g_cnt_min/ low_g_cnt_min value, the counter is cleared or decremented according to what is configured in the DBCNT bit of the configuration register.

The following figure shows the described behavior.

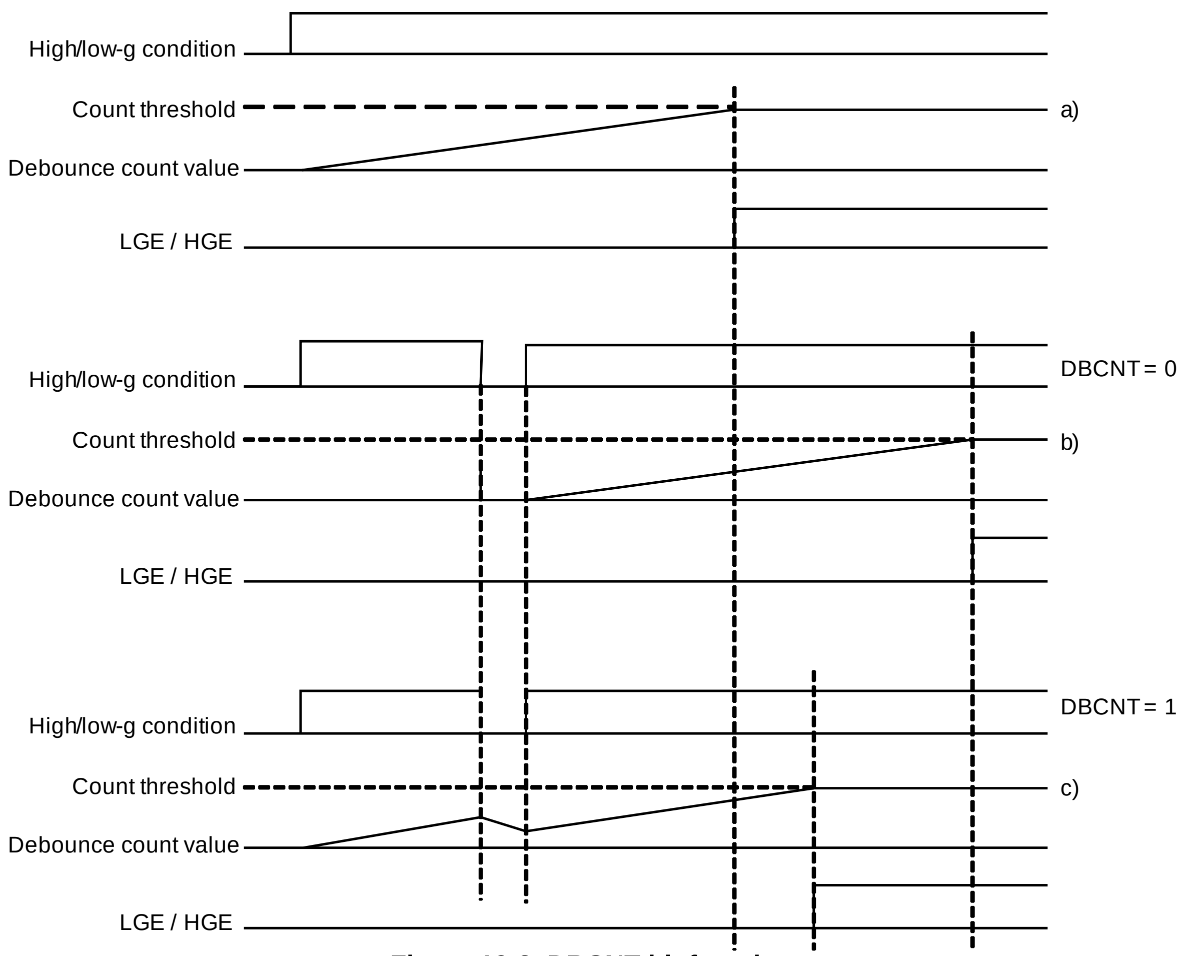


Figure 10-2. DBCNT bit function

For the AND flag to be set, the application must detect all the enabled axes as a low-g signal or as a high-g signal at the same time. If this condition is not met, the AND flag will remain cleared even when one or more single-axis flags are set.

For the OR flag to be set, the application must detect at least one of the enabled axes as a low-g or high-g signal.

The **HGOAE/LGOAE** bits allow the selection between the logical OR or AND of the low-g/high-g X, Y, and Z signals.

The threshold registers are **low_g_thresh/high_g_thresh**. The values from these registers are used to determine if the raw data signal has met the conditions to be interpreted as a high-g/low-g signal.

The following figure shows the region where the signal is interpreted as a low-g and high-g signal.

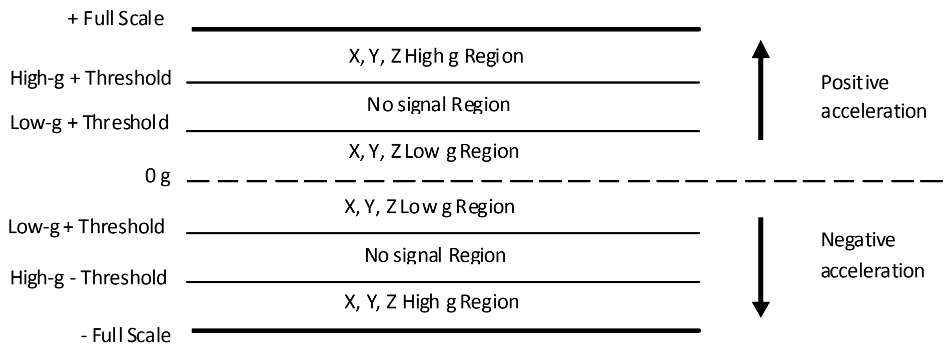


Figure 10-3. Low-g/High-g thresholds

10.2 High-g/Low-g configuration registers

10.2.1 low_g_thresh

Table 10-1. low_g_thresh registers

Offset	0x00(MSB)								0x01(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	low_g_thresh								low_g_thresh							
Reset	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0

Table 10-2. low_g_thresh bit description

Field	Description
low_g_thresh	A 16-bit element that indicates the low-g threshold. Any measurement below the low-g threshold will be considered as a low-g event. This value applies to all three axes. The reset value is 0x3E8 (decimal, 1000). Units: Gravity units (1 ADC count = 244 µg). Range of valid values: 1 (244 µg) to 32,767 (0x7FFF) for 8g.

10.2.2 low_g_cnt_min

Table 10-3. low_g_cnt_min registers

Offset	0x02							
Bit	7	6	5	4	3	2	1	0
Field	low_g_cnt_min							
Reset	0	0	0	0	1	0	0	0

Table 10-4. low_g_cnt_min bit description

Field	Description
low_g_cnt_min	An eight-bit number that sets the time that a signal must be kept below the low-g threshold in order to be considered as a low-g event. This value applies to all three axes. Units: Algorithm cycles (Time = [low_g_cnt_min] * [1/SR$_{LGHG}$]). Range of valid values: from 1 to 255. Reset value is 0x08, which corresponds to about 33 ms (8 x 1/244 Hz).

10.2.3 low_g_cfg

Table 10-5. low_g_cfg registers

Offset	0x03							
Bit	7	6	5	4	3	2	1	0
Field	Reserved			DBCNT	LGOAE	LGZEE	LGYEE	LGXEE
Reset	0x00			1	0	1	1	1

Table 10-6. low_g_cfg bit descriptions

Field	Description
DBCNT	Determines whether the debounce counter will be decremented or cleared. Range of valid values: from 0 to 1. • 0: Clear the debounce counter. • 1: Decrement the debounce counter.
LGOAE	Sets whether the low-g outputs will be ORed or ANDed. Range of valid values: from 0 to 1. • 0: Selects logical AND combination of low-g X, Y, Z axes event flag. • 1: Selects logical OR combination of low-g X, Y, Z axes event flag.
LGZEE	Enables or disables the low-g sensing for the Z axis. Range of valid values: from 0 to 1. • 0: Low-g event on Z axis disable. • 1: Low-g event on Z axis enable.
LGYEE	Enables or disables the low-g sensing for the Y axis. Range of valid values: from 0 to 1. • 0: Disables low-g event on Y axis. • 1: Enables low-g event on Y axis.
LGXEE	Enables or disables the low-g sensing for the X axis. Range of valid values: from 0 to 1. • 0: Disables low-g event on X axis. • 1: Enables low-g event on X axis.

0g detection example

The device powers up in Sleep mode, but it may become necessary to wake the part. This process uses the following mailboxes:

MB0 0x12 App_ID = 0x12; Power Controller modes

MB1 0x20 Command 0x2 = Write configuration; Offset = 0b

MB2 0x06 Offset = 0x06

MB3 0x01 Count of data to write

MB4 0x00 Actual Data Value: Clears the Sleep bit

Bytes to Send: 0x12, 0x20, 0x06, 0x01, 0x00.

Example 10-1.

To configure the MMA955xL platform to detect 0g type events, write the following transaction to the High-g/Low-g application:

1. MB0: Set APP_ID to (0x09).

   Selects the Low-g application.
2. MB1: Command to (0x20).

   Sends a Write Configuration space command.
3. MB2: Set offset to zero (0x00).

   Starts writing configuration values at register 0.
4. MB3: Set count field to (0x04).
5. MB3: Send Data (0x03).

   Sends 0x03E8, the 1000 decimal for the acceleration threshold.
6. MB3: Send Data (0xE8).
7. MB3: Send Data (0x08).

   Specifies 33 ms.
8. MB3: Send Data (0x17).

Bytes to Send: 0x09, 0x20, 0x00, 0x04, 0x03, 0xE8, 0x08, 0x17.

To determine if the low-g event happened, read the Low g application’s status register. Send the following command to set up for reading the status register.

Example 10-2.

1. MB0: Set APP_ID to (0x09).

   Selects the Low-g application.
2. MB1: Command to (0x30).

   Sends a Write Configuration space command
3. MB2: Set offset to zero (0x00).

   Starts writing configuration values at register 0.
4. MB3: Set count field to (0x01).

Bytes to Send: 0x09, 0x30, 0x00, 0x01.

Keeping the device flat on the desk, note that only X and Y should report a low-g event. That is indicated by the two lower bits being set in the status register.

Now read the mailboxes. The expected response to these commands is:

```
09 80 01 01 03
```

MB0: APP_ID = 0x09.

MB1: STATUS = 0x80.

Command Complete, no errors.

MB2: RequestedData count = 0x01.

MB3: Actual Data Count = 0x01.

MB4: Data = 0x03.

Only X and Y are showing a Low-g event.

At this point, the evaluation board was tossed up, so that it would go through a 0g event.

Request the status data by sending the following command to set up for reading the status register.

Example 10-3.

1. MB0: Set APP_ID to (0x09).
   Selects the Low-g application.
2. MB1: Command to (0x30).
   Sends a Write Configuration space command.
3. MB2: Set offset to zero (0x00).
   Starts writing configuration values at register 0.
4. MB3: Set count field to (0x01).

Bytes to Send: 0x09, 0x30, 0x00, 0x01.

Read the mailboxes. The expected response to these commands is:

```
09 80 01 01 0F
```

MB0: APP_ID = 0x09.

MB1: STATUS = 0x80.

Command Complete, no errors.

MB2: RequestedData count = 0x01.

MB3: Actual Data Count= 0x01.

MB4: Data = 0x0F.

All axes—X, Y, and Z—and/or flags are set.

10.2.4 high_g_thresh

Table 10-7. high_g_thresh registers

Offset	0x04(MSB)								0x05(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	high_g_thresh								high_g_thresh							
Reset	0	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0

Table 10-8. high_g_thresh bit description

Field	Description
high_g_thresh	A 16-bit element that indicates the high-g threshold. Any signal above the high-g threshold will be considered a high-g signal. This value applies to all three axes. Units: Gravity units (1 ADC count = 244 µg). Range of valid values: 1 (244 µg) to 32,767 (0x7FFF) for 8g.

10.2.5 high_g_cnt_min

Table 10-9. high_g_cnt_min register

Offset	0x06							
Bit	7	6	5	4	3	2	1	0
Field	high_g_cnt_min							
Reset	0	0	0	0	1	0	0	0

Table 10-10. high_g_cnt_min bit description

Field	Description
high_g_cnt_min	An eight-bit element that configures the time that a signal must be kept above the high-g threshold in order to be considered as a high-g signal. Units: Algorithm cycles (Time = [Algorithm cycles] * [1/SR$_{LGHG}$]). Range of valid values: 1 to 255.

10.2.6 high_g_cfg

Table 10-11. high_g_cfg register

Offset	0x07							
Bit	7	6	5	4	3	2	1	0
Field	Reserved			DBCNT	HGOAE	HGZEE	HGYEE	HGXEE
Reset	0x00			1	1	1	1	1

Table 10-12. high_g_cfg bit descriptions

Field	Description
Reserved	Bit field reserved.
DBCNT	Determines if the debounce counter will be decremented or cleared. Range of valid values: from 0 to 1. • 0: Clears the debounce counter. • 1: Decrements the debounce counter.
HGOAE	Sets whether the high-g outputs will be ORed or ANDed. Range of valid values: from 0 to 1. • 0: Enables the logical AND combination of the high-g X, Y, and Z axes. • 1: Enables the logical OR combination of high-g X, Y, and Z axes.
HGZEE	Enables or disables the high-g sensing for the Z axis. Range of valid values: from 0 to 1. • 0: Disables high-g event on Z axis. • 1: Enables high-g event on Z axis.
HGYEE	Enables or disables the high-g sensing for the Y axis. Range of valid values: from 0 to 1. • 0: Disables the high-g event on the Y axis. • 1: Enables the high-g event on the Y axis.
HGXEE	Enables or disables the high-g sensing for the X axis. Range of valid values: from 0 to 1. • 0: Disables the high-g event on the X axis. • 1: Enables the high-g event on the X axis.

10.2.7 lhg_event_mask

Table 10-13. lhg_event_mask register

Offset	0x09							
Bit	7	6	5	4	3	2	1	0
Field	FIFOHGE	—	—	—	FIFOLGE	—	—	—
Reset	0	0x00			0	0x00		

Table 10-14. lhg_event_mask bit descriptions

Field	Description
FIFOHGE	Pushes the high-g events into the Event queue. Range of valid values: from 0 to 1. • 0: Disables the push high-g events to the Event queue. • 1: Enables the push high-g events to the Event queue.
FIFOLGE	Pushes the low-g events into the Event queue. Range of valid values: from 0 to 1. • 0: Disables the push low-g events to the Event queue. • 1: Enables the push low-g events to the Event queue.

10.3 High-g/Low-g status register

10.3.1 lhg_out

Table 10-15. lhg_out register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	HGE	HGZ	HGY	HGX	LGE	LGZ	LGY	LGX
Reset	0	0	0	0	0	0	0	0

Table 10-16. lhg_out bit descriptions

Field	Description
HGE	Indicates the detection of logical AND/OR of all the enabled high-g events. Range of valid values: • 0: No high-g AND/OR event detected. • 1: A high-g AND/OR event detected.
HGZ	Indicates the detection of high-g signals on the Z axis. Range of valid values: • 0: No high-g event detected on the Z axis. • 1: A high-g event detected on the Z axis.
HGY	Indicates the detection of high-g signals on the Y axis. Range of valid values: • 0: No high-g event detected on the Y axis. • 1: A high-g event detected on the Y axis.
HGX	Indicates the detection of high-g signals on the X axis. Range of valid values: • 0: No high-g event detected on the X axis. • 1: A high-g event detected on the X axis.
LGE	Indicates the detection of logical AND/OR low-g events. Range of valid values: • 0: No low-g AND/OR events detected. • 1: A low-g AND/OR events detected.
LGZ	Indicates the detection of low-g signals on the Z axis. Range of valid values: • 0: No low-g event detected on the Z axis. • 1: A low-g event detected on the Z axis.
LGY	Indicates the detection of low-g signals on the Y axis. Range of valid values: • 0: No low-g event detected on the Y axis • 1: A low-g event detected on the Y axis.
LGX	Indicates the detection of low-g signals on the X axis. Range of valid values: • 0: No low-g event detected on the X axis • 1: A low-g event detected on the X axis.

Chapter 11 Tap Detection Application

11.1 Overview of Tap Detection application

Application ID	0x0A
Default speed	488 Hz
Configuration registers	Start on page 120.
Status registers	Start on page 124.

This application detects a “tap” or a “double-tap” event on any of the three accelerometer axes (X, Y, or Z). The result is reported independently along with the direction of the tap, positive or negative.

A “tap” is defined as an accelerometer movement that exceeds a user-defined magnitude threshold for a time falling between a user-specified minimum and maximum duration. A “double-tap” is two consecutive taps occurring in a user-defined period of time.

The acceleration data in the different ranges (2g, 4g, or 8g) is scaled or normalized to ±8g full scale, to ensure that the tap detection application will work the same in all ranges.

This algorithm works best with 8-g and 4-g, front-end configurations. This is because a 2-g configuration could produce saturated accelerometer values and report unexpected event or direction results.

Most taps have a magnitude greater than 0.75g and last for about 0.05 seconds. Double-taps generally are spaced at 0.2 seconds.

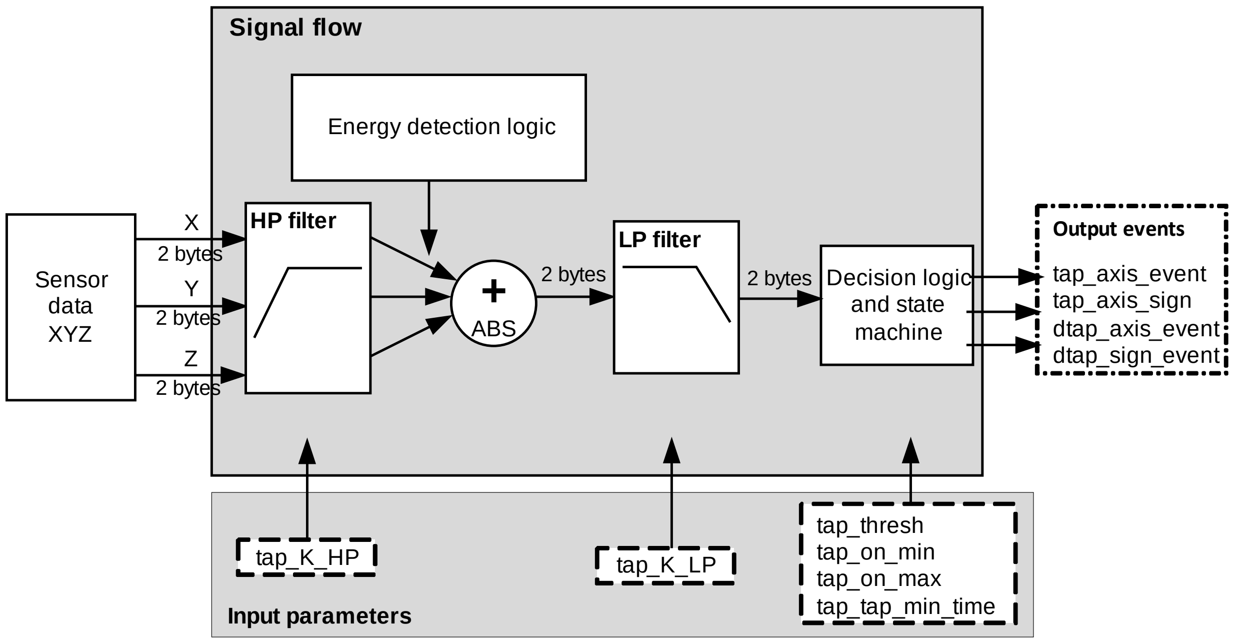
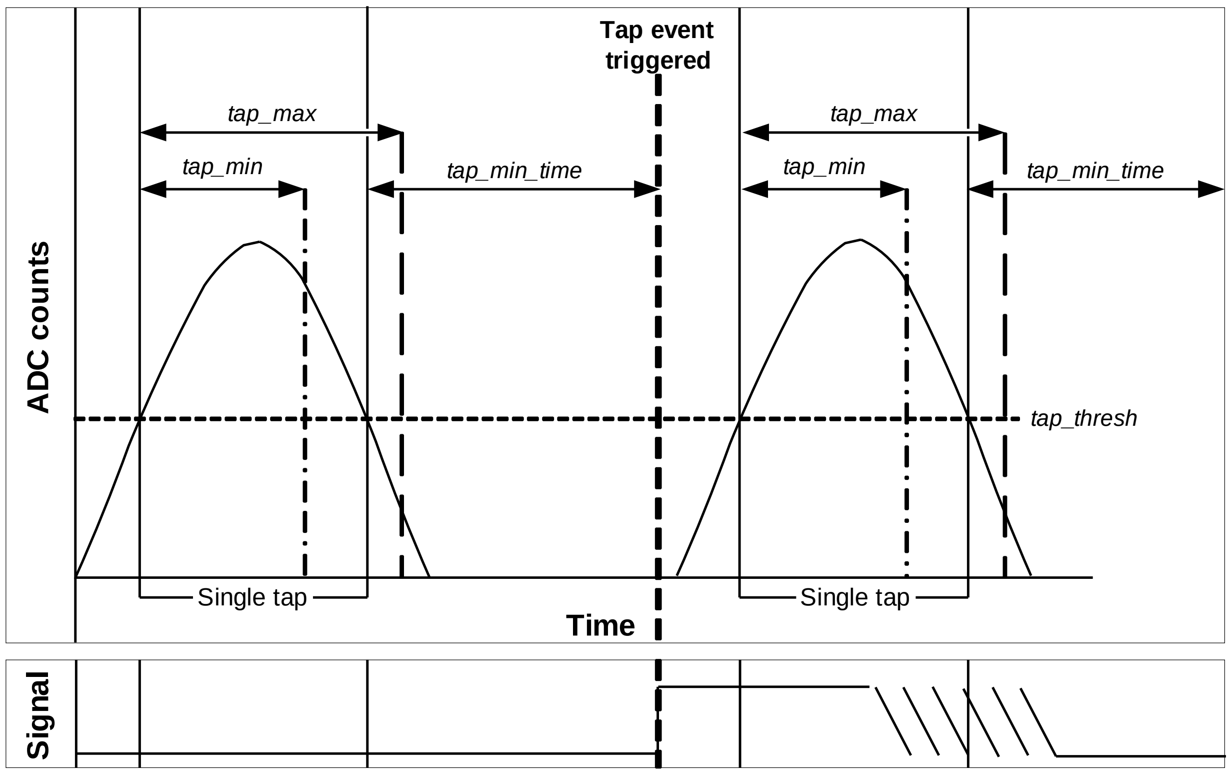


Figure 11-1. Tap detection application signal flow

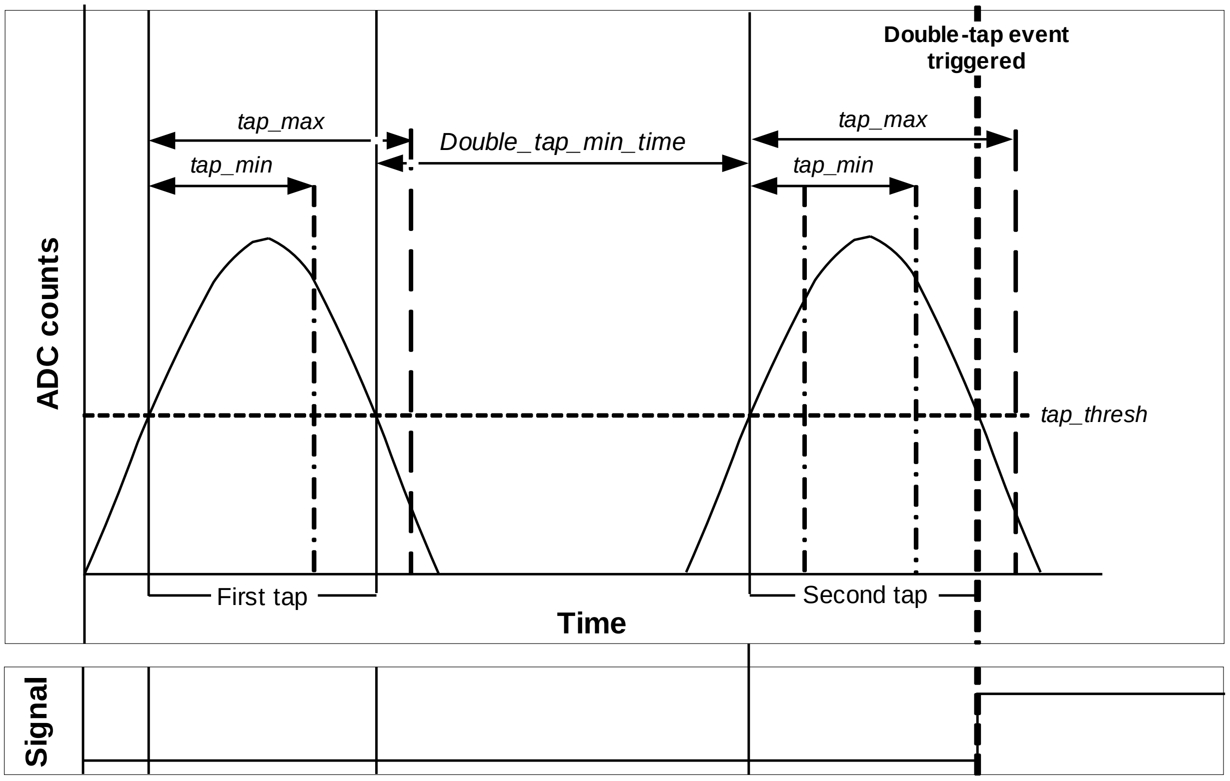


Legend:
- **Bold** text: Triggered events.
- *Italicized* text: Configuration settings.

Figure 11-2. Two single-tap detection sequence

When the signal coming from the LP filter crosses above the **tap_thresh** and continues higher—for more time than **tap_on_min** and less time than **tap_on_max**—the application captures the tap event. Subsequently, the signal must be lower than the **tap_thresh** until the **double_tap_min_**time is reached.

When all of these conditions occur, the tap detection event is triggered and reported.



Legend:
- **Bold** text: Triggered events.
- *Italicized* text: Configuration settings.

Figure 11-3. Double-tap detection sequence

For double-tap detection, the sequence is almost identical. Just after the first tap has been detected, however, and before the **double_tap_min_time** has expired, a second input signal rises above the **tap_thresh** magnitude. The application evaluates the second tap to determine if it is valid or not. If both taps are valid, the algorithm reports a double-tap event instead of two separate taps.

For more information, see *MMA8450Q Single/Double and Directional Tap Detection Application Note* (AN3919).

The **double_tap_min_time** time determines how quickly the second tap must occur to qualify as a double-tap event.

11.2 Tap-Detection configuration registers

11.2.1 tap_thresh

Table 11-1. tap_thresh registers

Offset	0x00(MSB)								0x01(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	tap_thresh								tap_thresh							
Reset	0x0B								0x0B							

Table 11-2. tap_thresh bit description

Field	Description
tap_thresh [7:0]	Sets the minimum magnitude that the signal must reach to be detected as a tap/double-tap event. Units: ADC Counts. (1 ADC count = 244 µg.) Range of valid values: -32,767 to 32,767 (that is, $\underline{+}$ 8g). Reset value: 0x0B0B, decimal 2827 or about 0.69g.

11.2.2 tap_on_min

Table 11-3. tap_on_min register

Offset	0x02							
Bit	7	6	5	4	3	2	1	0
Field	tap_on_min							
Reset	0x0A							

Table 11-4. tap_on_min bit description

Field	Description
tap_on_min [7:0]	Sets the minimum time that the signal must remain above the threshold to be considered a tap event. Units: Algorithm cycles (Time = [Algorithm cycles] * [1/SR_{TAP}]). Range of valid values: 1 to 255.

11.2.3 tap_on_max

Table 11-5. tap_on_max register

Offset	0x03							
Bit	7	6	5	4	3	2	1	0
Field	tap_on_max							
Reset	0x78							

Table 11-6. tap_on_max bit description

Field	Description
tap_on_max [7:0]	Sets the maximum time that the signal must remain above the threshold in order to consider a tap event. Units: Algorithm cycles (Time = [Algorithm cycles] * [1/SR_{TAP}]). Range of valid values: 1 to 255.

11.2.4 double_tap_min_time

Table 11-7. double_tap_min_time register

Offset	0x04							
Bit	7	6	5	4	3	2	1	0
Field	double_tap_min_time							
Reset	0xFF							

Table 11-8. double_tap_min_time bit description

Field	Description
double_tap_min_ti me [7:0]	Sets the time required between taps to qualify as a double-tap, rather than two separate taps. If the second tap occurs before the defined time, the event is considered a double-tap. Units: Application cycles (Time = [Algorithm cycles] * [1/SR_{TAP}]). Range of valid values: 1 to 255.

11.2.5 tap_K_HP

Table 11-9. tap_K_HP register

Offset	0x05							
Bit	7	6	5	4	3	2	1	0
Field	tap_K_HP							
Reset	0x05							

Table 11-10. tap_K_HP bit description

Field	Description
tap_K_HP [7:0]	High-pass filter’s configurable cutoff. For high-pass filter K values, see Table 8-4 on page 82. Units: None. Range of valid values: 0 to 15. (Value of zero means no filter.)

11.2.6 tap_K_LP

Table 11-11. tap_K_LP register

Offset	0x06							
Bit	7	6	5	4	3	2	1	0
Field	tap_K_LP							
Reset	0x04							

Table 11-12. tap_K_LP bit description

Field	Description
tap_K_LP [7:0]	Low-pass filter’s configurable cutoff. For high-pass filter K values, see Table 8-3 on page 81. Units: None. Range of valid values: 0 to 15. (Value of zero means no filter.)

11.2.7 tap_axis_enable

Table 11-13. tap_axis_enable register

Offset	0x07							
Bit	7	6	5	4	3	2	1	0
Field	Reserved					Z_enable	Y_enable	X_enable
Reset	0x00					1	1	1

Table 11-14. tap_axis_enable bit descriptions

Field	Description
Reserved [7:3]	Bit field reserved.
Z_enable Y_enable X_enable [2:0]	Enables or disables detection of each axis. Units: Non-dimensional. Range of valid values: 0 to 255. • 0: Disables the axis. • 1: Enables the axis.

11.2.8 tap_events_mask

Table 11-15. tap_events_mask registers

Offset	0x08							
Bit	7	6	5	4	3	2	1	0
Field	Reserved							Tap/ DTap mask
Reset	0x00							0

Table 11-16. tap_events_mask bit description

Field	Description
tap_events_mask	Masks a specific event of the application to be pushed into the event queue. Units: Non-dimensional. Range of valid values: 0 to 255. • 0: Disables event to be pushed into the event queue. • 1: Enables event to be pushed into the event queue.

11.3 Tap-Detection status registers

11.3.1 TAP_Out

Table 11-17. TAP_Out register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	TAP	—	Z Dir	Z Ev	Y Dir	Y Ev	X Dir	X Ev
Reset	0	0	0	0	0	0	0	0

Table 11-18. TAP_Out bit description

Field	Description
TAP	Reports whether a tap event has been detected. • 0: Event not detected. • 1: Event detected.
Z Dir	Indicates the direction of the Z-axis event. • 0: Event in positive direction. • 1: Event in negative direction.
Z Ev	Reports whether a Z-Axis event has been detected. • 0: Event not detected. • 1: Event detected.
Y Dir	Indicates the direction of the Y-axis event. • 0: Event in positive direction. • 1: Event in negative direction.
Y Ev	Reports whether a X-Axis event has been detected. • 0: Event not detected. • 1: Event detected.
X Dir	Indicates the direction of the X-axis event. • 0: Event in positive direction. • 1: Event in negative direction.
X Ev	Reports whether a X-Axis event has been detected. • 0: Event not detected. • 1: Event detected.

11.3.2 DTAP_Out

Table 11-19. DTAP_Out register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	DTAP	—	Z Dir	Z Ev	Y Dir	Y Ev	X Dir	X Ev
Reset	0	0	0	0	0	0	0	0

Table 11-20. DTAP_Out bit description

Field	Description
DTAP	Reports whether a double-tap event has been detected. • 0: Event not detected. • 1: Event detected.
Z Dir	Indicates the direction of the Z-axis event. • 0: Event in positive direction. • 1: Event in negative direction.
Z Ev	Reports whether a Z-Axis event has been detected. • 0: Event not detected. • 1: Event detected.
Y Dir	Indicates the direction of the Y-axis event. • 0: Event in positive direction. • 1: Event in negative direction.
Y Ev	Reports whether a X-Axis event has been detected. • 0: Event not detected. • 1: Event detected.
X Dir	Indicates the direction of the X-axis event. • 0: Event in positive direction. • 1: Event in negative direction.
X Ev	Reports whether a X-Axis event has been detected. • 0: Event not detected. • 1: Event detected.

Chapter 12 Tilt Application

12.1 Overview of Tilt application

Application ID	0x0B
Default speed	122 Hz
Configuration registers	Start on page 129.
Status registers	Start on page 132.

This application senses the angle between the Y/Z and X/Z planes with the fixed reference to gravity. The application also determines the quadrant of each of the enabled planes, referenced to gravity.

The application collects the three-axis data from the sensor and passes the accelerometer data through a low-pass filter with a user-configurable cut-off frequency. The filter output data determines the quadrant and the tilt angle.

The following figure shows the tilt application block diagram. All the outputs referring to the X/Y plane always read zero due to the ambiguous relationship between that plane and the axes.

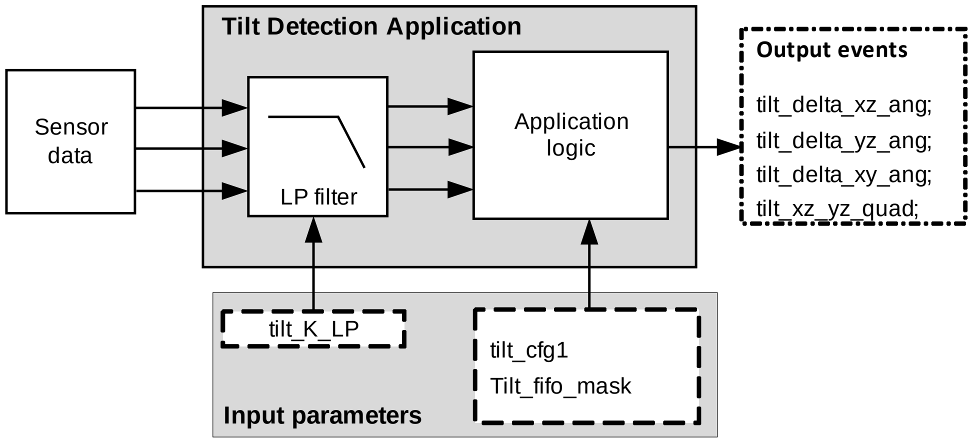


Figure 12-1. Tilt application signal flow

The following figure shows the X/Z and Y/Z tilt angle outputs, relative to the fixed ground reference plane.

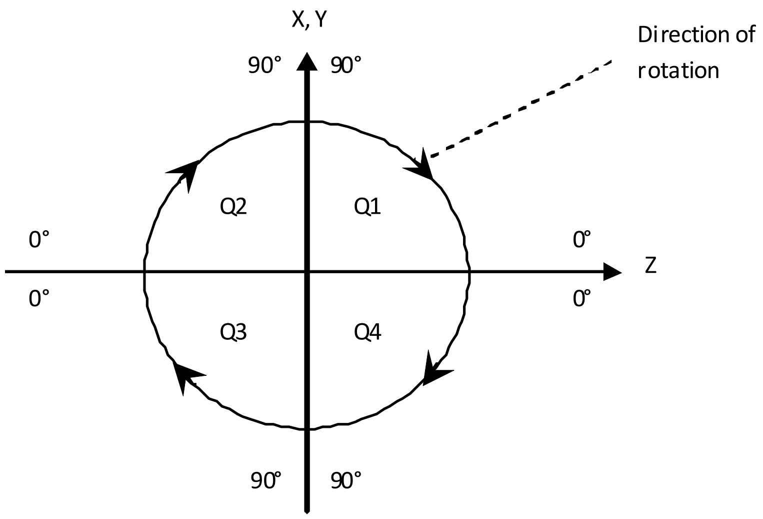


Figure 12-2. X/Z and Y/Z tilt angle outputs, relative to fixed ground

The following table shows the quadrant definition for planes X/Z and Y/Z, depending on the sign of the axis signals.

Table 12-1. X/Z and Y/Z quadrant definitions

Quadrant	X/Y acceleration	Z acceleration
1	+	+
2	+	-
3	-	-
4	-	+

12.2 Tilt configuration registers

12.2.1 tilt_K_LP

Table 12-2. tilt_K_LP register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	tilt_K_LP							
Reset	0	0	0	0	0	1	0	1

Table 12-3. tilt_K_LP bit description

Field	Description
tilt_K_LP	This element and the sample rate define the bandwidth of the low-pass filter. For the relationship between the K constant and the cut-off frequency, see Table 4-5 on page 43 and Table 4-10 on page 51. The cutoff frequency is given at -3 dB. Units: None. Range of valid values: 0 to 7. Reset: Values to 0x05g.

12.2.2 tilt_cfg1

Table 12-4. tilt_cfg1 register

Offset	0x01							
Bit	7	6	5	4	3	2	1	0
Field	Reserved	XYEN	YZEN	XZEN	Angle_thresh [3:0]			
Reset	0	1	1	1	1	0	0	1

Table 12-5. tilt_cfg1 bit descriptions

Field	Description
Reserved	Bit field reserved.
XYEN	Enables or disables the sensing of the X/Yplane. Range of valid values: from 0 to 1. • 0: Disables X/Y-plane sensing angle. • 1: Enables X/Y-plane sensing angle.
YZEN	Enables or disables the sensing of the Y/Z plane. Range of valid values: from 0 to 1. • 0: Disables Y/Z-plane sensing angle. • 1: Enables Y/Z-plane sensing angle.
XZEN	Enables or disables the sensing of the X/Z plane. Range of valid values: from 0 to 1. • 0: Disables X/Z-plane sensing angle. • 1: Enables X/Z-plane sensing angle.
Angle_thresh	Configures a delta angle threshold. Whenever the algorithm detects a delta angle variation above the established threshold, from one execution to the next, the angle flag is set. Units: Degrees. Range of valid values: 1 to 10.

12.2.3 tilt_event_mask

Table 12-6. tilt_event_mask register

Offset	0x02							
Bit	7	6	5	4	3	2	1	0
Field	Quad_Change	Reserved						
Reset	0	0x00						

Table 12-7. tilt_event_mask bit descriptions

Field	Description
Quad_Change	Configures the event that will be pushed into the Event Queue Application. Currently there is one event available. If enabled, the application will push the new quadrant byte into the Event Queue every time the quadrant information changes. Range of valid values: 0 to 1. • 0: Disables events being added to the Event Queue. • 1: Enables tilt events to be added to the Event Queue.
Reserved	Bit field reserved.

12.3 Tilt status registers

12.3.1 tilt_delta_xz_ang

Table 12-8. tilt_delta_xz_ang register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	ANGFLG	XZANGLE [6:0]						
Reset	0	0x00						

Table 12-9. tilt_delta_xz_ang bit descriptions

Field	Description
ANGFLG	Stores the angle flag that is set whenever the difference between the current angle and the last-calculated angle is bigger than the Angle_thresh. The computation uses an approximation of the atan formula ≥ within ± 1° of accuracy only. Units: Flag. Range of valid values: from 0 to 1. • 0: Delta angle is not above the threshold. • 1: Delta angle went above the threshold.
XZANGLE	Stores the tilt angle of the X and Z axes. Units: Degrees. Range of valid values: 0 to 90.

12.3.2 tilt_delta_yz_ang

Table 12-10. tilt_delta_yz_ang register

Offset	0x01							
Bit	7	6	5	4	3	2	1	0
Field	ANGFLG	YZANGLE[6:0]						
Reset	0	0x00						

Table 12-11. tilt_delta_yz_ang bit descriptions

Field	Description
ANGFLG	Stores the angle flag that is set whenever the difference from the current angle with the last-calculated angle is bigger than the configured threshold. Units: Flag. Range of valid values: 0 to 1. • 0: No occurrence of delta angle above threshold. • 1: Delta angle went above threshold.
YZANGLE	Stores the tilt angle of Y and Z axes. Units: Degrees. Range of valid values: 0 to 90.

12.3.3 tilt_delta_xy_ang

Table 12-12. tilt_delta_xy_ang register

Offset	0x02							
Bit	7	6	5	4	3	2	1	0
Field	ANGFLG	XYANGLE [6:0]						
Reset	0	0x00						

Table 12-13. tilt_delta_xy_ang bit descriptions

Field	Description
ANGFLG	Disabled by default and always reads zero. Units: Flag. Range of valid values: Always reads zero.
XYANGLE	Disabled by default and always reads zero. Units: Degrees. Range of valid values: Always reads zero.

12.3.4 tilt_xz_yz_quad

Table 12-14. tilt_xz_yz_quad register

Offset	0x03							
Bit	7	6	5	4	3	2	1	0
Field	QUADFLAG	UNUSED	XZ_QUAD		YZ_QUAD		XY_QUAD	
Reset	0	0	0x00		0x00		0x00	

Table 12-15. tilt_xz_yz_quad bit descriptions

Field	**Description**
QUADFLAG	Records when a change in quadrant is detected. Units: Flag. Range of valid values: • 0: No change detected. • 1: Change detected.
XZ_QUAD [5:4]	Stores the quadrant change detected between X and Z axes. Units: Quadrant number. Range of valid values: • 00: Quadrant 1. • 01: Quadrant 2. • 10: Quadrant 3. • 11: Quadrant 4.
YZ_QUAD [3:2]	Stores the quadrant detected between Y and Z axes. Units: Quadrant number. Range of valid values: • 00: Quadrant 1. • 01: Quadrant 2. • 10: Quadrant 3. • 11: Quadrant 4.
XY_QUAD [1:0]	Stores the quadrant-change detected between the X and Y axes. This information is disabled by default. Units: Quadrant number. Range of valid values: Always reads zero.

Chapter 13 Frame Counter Application

Application ID	0x0E
Default speed	488 Hz
Configuration registers	None.
Status registers	Start on page 135.

13.1 Overview of Frame Counter application

The Frame Counter application reports the number of frames that have been processed at the application's execution rate.

The application counts frames, not time. It is not real-time because the device can be put into a Stop mode where time passes, but no samples are taken. It does not count when the device is in stop mode.

13.2 Frame-Counter configuration registers

There are no configuration registers for this application.

13.3 Frame-Counter status register

13.3.1 frame_cnt

Table 13-1. frame_cnt registers

Offset	0x00(MSB)								0x01(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	frame_cnt															
Reset	0x00															

Table 13-2. frame_cnt bit description

Field	Description
frame_cnt	Counter with the total of frames at 488 Hz. Units: Non-dimensional. Range of valid values: 0 to 65535.

Chapter 14 Data FIFO Application

14.1 Overview of Data FIFO application

Application ID	0x0F
Default speed	488 Hz
Configuration registers	Start on page 145.
Status registers	Start on page 148.

The Data FIFO (First In First Out) application is a buffer intended to store the output data from an application. Every scheduler interval, the Data FIFO application gathers the output data from an application and stores it until the host processor reads the data.

The FIFO application uses the mailboxes differently than other applications—operating in the streaming mode. In streaming mode, the host continues reading data until a maximum of up to 255 bytes is read per host request. The host must read all the data that it requested.

The FIFO can be connected to one application and can collect packets of data in different sizes (one, two, four, or six bytes).

The Data FIFO has two different modes of operation: Stop on Overflow and Free Run.

The application’s status register displays current status values such as overflow condition, watermark reached, and buffer empty. These conditions have a flag that is asserted individually each time one of these conditions occurs.

The user can configure the FIFO buffer size by writing to the fifo_size word within the configuration registers. The buffer size can be configured only once and is limited by the amount of RAM available. The amount of available RAM can be impacted by the Event Queue Application which also can be configured to use large amounts of RAM.

NOTE

Before configuring the Data FIFO application, it is recommended that the Data FIFO Application be suspended. After the parameter is configured, remove the application from suspend. A suspend is done with the Reset/Suspend/Clear Application (APP_ID 0x17).

14.2 Modes of operation

This section examines the modes of operation of the FIFO buffer.

14.2.1 Stop-on-overflow

The FIFO stores data from an application, every single frame (a scheduler interval), as long as the FIFO is enabled. In this mode, the FIFO stores data until an overflow condition is reached. At an overflow, the overflow flag is set.

The host asynchronously reads payload packets from the Data FIFO application. Reading payload packets frees up slots for new entries into the FIFO. If the host reads data faster than the applications put data into the FIFO, the overflow condition will never happen.

The overflow condition occurs when the available buffer memory is full and there is no space available for another packet.

14.2.2 Free-run

The FIFO behaves as a circular buffer that stores data from the configured channel, every single frame, as long as the FIFO is enabled. In this mode, the FIFO never stops storing data, even though the overflow condition is reached and the overflow flag is set.

When the FIFO becomes full, the oldest data in the buffer will be overwritten first.

14.3 Reading process

To read the data FIFO, the host sends a read-data command to the MMA955xL device that calls the Data FIFO application, stores the requested data within the FIFO into a buffer, and returns the number of bytes read. The data FIFO does not tag each entry with a timestamp, only storing a timestamp for the last entry.

When the reading process is performed, the Data FIFO application calculates the timestamp for the first group of requested data and appends the timestamp to the data. If the host requests *N* bytes and that is bigger than the entry size, the pop routine will append the timestamp only for the first entry that fits in the *N* bytes requested. The host must calculate the timestamps for the extra entries requested within the *N* bytes.

The host can request to read up to 255 bytes at a time. Assuming the AFE is providing data, this is exactly 36 packets of AFE data.

Each host, FIFO-read transaction includes a status byte and a timestamp word. These three bytes are prefixed to the payload data.

Payload data is prefixed with the APP_ID of the application that has provided the data. In the AFE case, the APP_ID is 0x06.

(255 - 3 (status and timestamp)) / 7 (APP_ID + 6 byte XYZ data) = 36

Reading the FIFO example

The device powers up in Sleep mode, but it may become necessary to wake the MMA955xL device.

Waking the device uses the following mailboxes:

MB0 0x12 App_ID = 0x12; Power Controller modes

MB1 0x20 Command 0x2 = Write configuration; Offset = 0

MB2 0x06 Offset = 0x06

MB3 0x01 Count of data to write

MB4 0x00 Actual Data Value; Clears sleep bit

Bytes to Send: 0x12, 0x20, 0x06, 0x01, 0x00.

Example 14-1.

To configure for FIFO operation, write the following transaction to the Data-FIFO application:

1. MB0: Set APP_ID to (0x0F).
   Selects the Data-FIFO application.
2. MB1: Command to (0x20).
   Command is a Write Configuration space.
3. MB2: Set offset to zero (0x00).
   Start writing configuration values at register 0.
4. MB3: Set count field to (0x0A).
   Sends nine bytes.
5. MB4: Send Data (0x0C).
   Sends six bytes, free-run mode.
6. MB5: Send Data (0x00).
7. MB6: Send Data (0x00).
8. MB7: Send Data (0x00).
9. MB8: Send Data (0x00).
10. MB6: Send Data (0x3C).
    Reserves 60 bytes—10 packets of six bytes each.
11. MB7: Send Data (0x06).
    Associates APP ID = AFE - .
12. MB5: Send Data (0x00).
13. MB6: Send Data (0x00).
14. MB7: Send Data (0x00).

Bytes to Send: 0x0F, 0x20, 0x00, 0x0A, 0x0C, 0x00, 0x00, 0x00, 0x00, 0x3C, 0x06, 0x00, 0x00, 0x00.

The configuration registers can only be written once. To reconfigure the FIFO, the device first must be reset.

Read the FIFO applications status register to get the data.

Send the following command to set up for reading the status register.

Example 14-2.

1. MB0: Set APP_ID to (0x0F).
   Selects the Data-FIFO application.
2. MB1: Command to (0x30).
   Sends a Read Status space command.

3. MB2: Set offset to zero (0x03).

   Starts reading the configuration values from register 0x03.
4. MB3: Set count field to (0x0A).

   Reads 10 bytes (three status bytes, one APP_ID, and six bytes of data).

Bytes to Send: 0x0F, 0x30, 0x03, 0x0A.

This results in the following response.

0F 80 0A 0A 05 83 13 06 00 81 FF EA 11 0C

The last six bytes are the AFE data: X, Y, and Z.

This response was with the device flat, in the Face-Up orientation. The X and Y data are close to 0x0000—or 0-g—and the Z data is close to 0x1000—or 1-g.

MB0: APP_ID = 0x0F; FIFO application

MB1: STATUS = 0x80; Command Complete, no errors

MB2: RequestedData count = 0x0A; 10 bytes of data

MB3: Actual Data Count = 0x0A; 10 bytes of data

MB4: 0x05 = FIFO Status; Watermark and Overflow flag is set

MB5-6: = 0x8313; Timestamp

MB7: APP ID of the application providing the actual data = 0x06; AFE application

MB8-9: AFE X data; 0x0081

MB10-11: AFE Y data; 0xFFEA

MB12-13: AFE Z Data; 0x110C

The read size for one (1+6 byte) payload is 10 bytes. The size for two payloads is 17 bytes, 3 = 24 bytes.

Reading three payloads

Send the following command to set up for reading the status register.

Example 14-3.

1. MB0: Set APP_ID to (0x0F).

   Selects the Low-g application.
2. MB1: Command to (0x30)

   Sends a Write Configuration space command.
3. MB2: Set offset to zero (0x03).

   Starts writing configuration values at register 0.
4. MB3: Set count field to (0x18)

   Reads 24 bytes (3 status byes + 3x(APP_ID+6 data)).

Bytes to Send: 0x0F, 0x30, 0x03, 0x18.

The response is below:

0F 80 18 18 05 33 53 06 00 93 00 7F 11 06 06 00 A2 00 7D 11 08 06 00 9C 00 79 11 07

The last six bytes are the AFE data, X, Y, and Z.

This response was with the device flat, in the Face-Up orientation. The X and Y data are close to 0x0000—or 0-g—and the Z data is close to 0x1000—or 1-g.

MB0: APP_ID = 0x0F; FIFO application

MB1: STATUS = 0x80; Command Complete, no errors

MB2: Requested data count = 0x18; 24 bytes of data

MB3: Actual Data Count = 0x18; 24 bytes of data

MB4: 0x05 = FIFO Status; Watermark and Overflow flag is set

MB5-6: = 0x3353; Timestamp

MB7: APP ID of the application providing the actual data = 0x06; The AFE application

MB8-9: AFE X data; 0x0093

MB10-11: AFE Y data; 0x007F

MB12-13: AFE Z Data; 0x1106

MB14: APP ID of the application providing the actual data = 0x06; the AFE application

MB15-16: AFE X data; 0x00A1

MB17-18: AFE Y data; 0x007D

MB19-20: AFE Z Data; 0x1108

MB21: APP ID of the application providing the actual data = 0x06; the AFE application

MB22-23: AFE X data; 0x009C

MB24-25: AFE Y data; 0x0079

MB26-27: AFE Z Data; 0x1107

To read data stored by the data FIFO application, the host must send a Read Data command along with a specific offset value.

This process has three conditions:

- The offset must be fixed to three
- The number of bytes to read must be larger than one and a multiple of the entry size
- The host must add three bytes to de-count the total bytes to read

It is extremely important to adhere to these three conditions or the data FIFO could enter into an error state and be unable to pop coherent data. To recover the data FIFO, reset the flag in the Reset_control byte (APP_ID 0x17).

When the command is sent, the first four bytes that the MMA955xL device returns correspond to the response of the command. The next byte gives the status of the data FIFO application module and the following two bytes represent the timestamp. Hence the three-byte offset.

These bytes are part of those requested by the host, so it is very important that the user add three bytes to the number of bytes to read. The data stored in the data FIFO comes after these seven bytes.

If the module is storing data at the exact moment that a read is issued by the host, the MMA955xL device will return an error condition (A2). When this occurs, the host must retry reading the data. If any other error is returned, the host must take the proper action.

14.4 Data FIFO block diagram

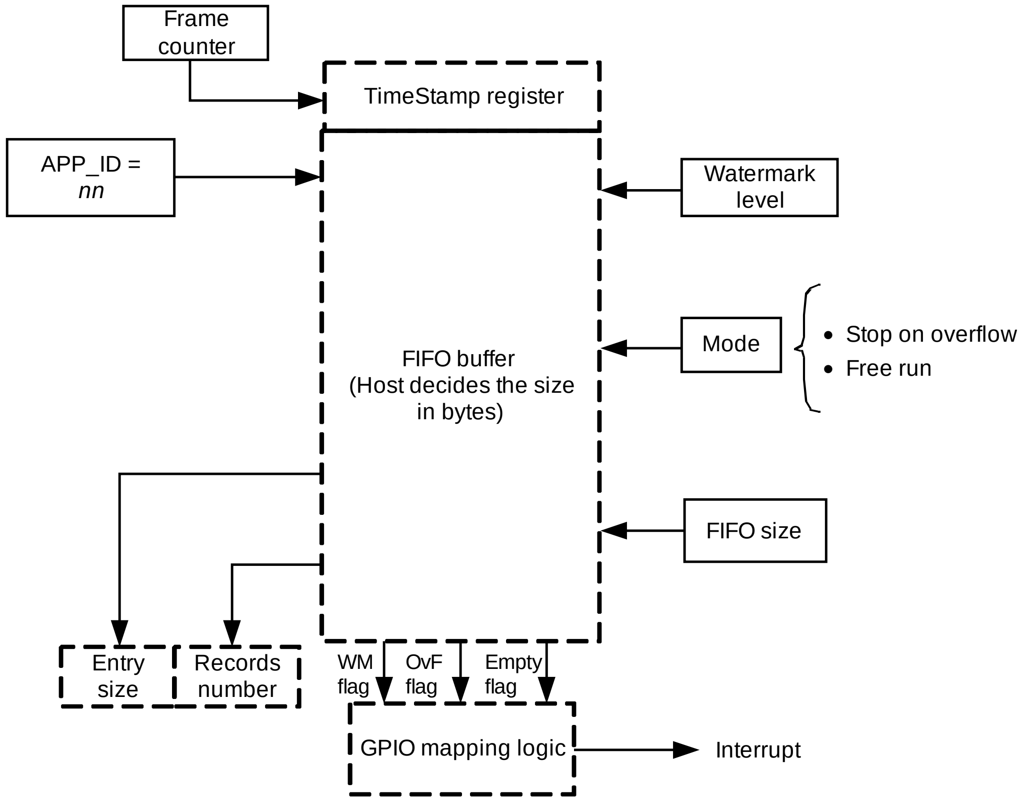


Figure 14-1. Data FIFO model

The preceding figure shows the block-level model of the data FIFO. The application is driven by the watermark and mode inputs that configure the functionality.

The watermark level helps the host prevent data loss by raising a warning just before the overflow condition occurs. When the number of bytes in the buffer approaches its capacity, the data FIFO asserts the watermark flag in the status register. The warning level is user-configurable. If the level of 0 is configured, the watermark is disabled.

The channel has its own data format code (DFC) bit field that configures the payload to be stored (one, two, four, or six bytes). A NULL or zero value in the APP_ID Channel register means the channel is disabled. Any valid APP_ID value means the channel is enabled.

The data FIFO has a status register in its output structure that contains status flags such as watermark, overflow, and buffer empty. These flags can be mapped to the GPIO pins to generate an interrupt to the host.

The output also has an Entry Size register that shows the size, in bytes, of each entry that the data FIFO calculated, according to its payload configuration. A Records Number register records the number of entries stored in the data FIFO. Both registers help the host to calculate how many bytes to request, so that it can dump the entire FIFO buffer. A simple multiplication of those two registers’ value gives the total number of bytes stored in the FIFO.

14.4.1 Entries format

Channel enabled

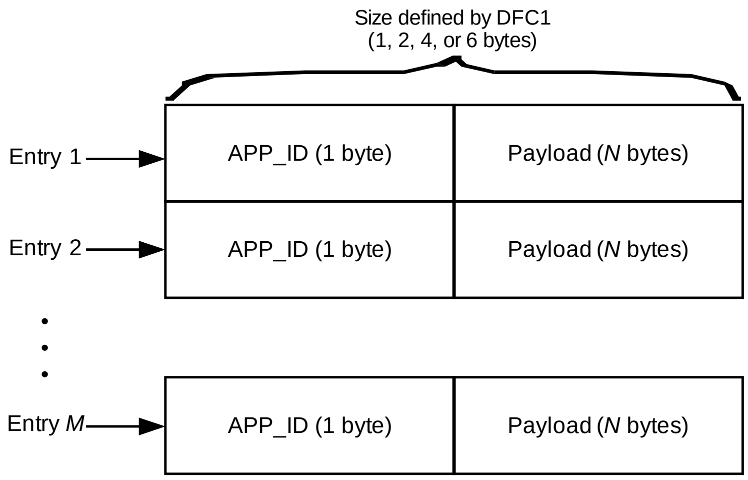


Figure 14-2. FIFO-entry formats, when channel enabled

14.5 Data FIFO configuration registers

This section contains the FIFO configuration registers. These registers can only be written once. To re-configure the FIFO, the device must be reset. This is because the FIFO application requests RAM and RAM can only be allocated one time.

14.5.1 FIFO configuration byte

Table 14-1. FIFO Config Byte register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	Reserved				DFC1		Mode	
Reset	0x00				0x00		0x00	

Table 14-2. FIFO Config Byte bit descriptions

Field	Description
Reserved	Bit field reserved.
DFC1 (Data Format Code)	Configures the payload size of the channel. Units: Non-dimensional. Range of valid values: 0 to 3. • 00: One byte. • 01: Two bytes. • 10: Four bytes • 11: Six bytes.
Mode	Configures the FIFO mode. Units: Non-dimensional. Range of valid values: • 00: Free Run. • 01: Stop On Overflow.

14.5.2 FIFO size word

Table 14-3. FIFO-size word registers

Offset	0x04(MSB)								0x05(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	FIFO Size Word 1								FIFO Size Word 2							
Reset	0x00								0x00							

Table 14-4. FIFO-size word bit description

Field	Description
FIFO Size Word	Reserves the maximum size (in bytes) that the FIFO can use. This is limited by the available RAM. Units: Bytes.. Range of valid values: 0 to available RAM. 0x1C8 (decimal, 456) is the maximum amount of RAM that can be requested from the MMA955xL device. The RAM is shared by multiple applications, so be careful not to set the FIFO to use more memory than is physically available. That will cause unknown and undesirable results.

14.5.3 FIFO APP_ID

Table 14-5. FIFO Channel APP_ID register

Offset	0x06							
Bit	7	6	5	4	3	2	1	0
Field	FIFO Ch APP_ID							
Reset	0x00							

Table 14-6. FIFO Channel APP_ID bit description

Field	Description
FIFO Ch APP_ID	The APP_ID of the application that supplied data to be buffered by the FIFO. Range of valid values: 0 to 0xFF.

14.5.4 Watermark

Table 14-7. Watermark registers

Offset	0x08(MSB)								0x09(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	Watermark								Watermark							
Reset	0x00								0x00							

Table 14-8. Watermark bit description

Field	Description
Watermark	Sets the count, in bytes, for the FIFO to set or clear the watermark flag. Units: Bytes.. Range of valid values: 0 - available RAM.

14.6 Data FIFO status registers

14.6.1 Records number

Table 14-9. Records-number register

Offset	0x00(MSB)								0x01(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	Records Number								Records Number							
Reset	0x00								0x00							

Table 14-10. Records-number bit description

Field	Description
Records Number	This word stores the current number of records in the FIFO buffer. A record is the payload packet of data comprised of an APP_ID and the actual data—which could be one, two, four, or six bytes long. A record can be two, three, four, or seven bytes long. Units: Number of entries in the Data FIFO (entry size in bytes = Ch APP_ID+ DFC1). Range of valid values: 0 to available oxFFFF.

14.6.2 Entry size

Table 14-11. Entry-size register

Offset	0x02							
Bit	7	6	5	4	3	2	1	0
Field	Entry Size							
Reset	0x00							

Table 14-12. FIFO config-byte/bit description

Field	Description
Entry Size	This byte shows the size in bytes of each entry that the data FIFO has stored, that calculation based on its configuration. Units: Bytes. Range of valid values: 0 to 6.

14.6.3 FIFO_Status

Table 14-13. FIFO_Status register

Offset	0x03							
Bit	7	6	5	4	3	2	1	0
Field	on_going_push	on_going_pop	Reserved			ovf_flag	empty_ flag	wmrk_ flag
Reset	0x00	0x00	0x00			0	0	0

Table 14-14. FIFO_Status bit descriptions

Field	Description
fifo_on_going_push	Indicates that a push operation is being executed—the FIFO receiving data from the application. Units: Non-dimensional. Range of valid values: 0 to 1. • 0: A push is not being executed. • 1: A push is being executed.
fifo_on_going_pop	Indicates that a pop operation is being executed. The FIFO is sending out data and it is being read. Units: Non-dimensional. Range of valid values: 0 to 1. • 0: A pop operation is not being executed. • 1: A pop operation is being executed.
Reserved	Bit field reserved.

Table 14-14. FIFO_Status bit descriptions (continued)

Field	Description
fifo_ovf_flag	Indicates that FIFO buffer has reached the maximum number of entries allowed. This meaning can vary depending of the mode of operation. Units: Non-dimensional. Range of valid values: • 0: Not overflow condition. • 1: Overflow condition.
fifo_empty_flag	Indicates whether the FIFO buffer is empty. Units: Non-dimensional. Range of valid values: • 0: FIFO buffer is not empty. • 1: FIFO buffer is empty.
fifo_wmrk_flag	Indicates when the entries in the FIFO buffer have reached the watermark count. Units: Non-dimensional.. Range of valid values: • 0: Watermark not reached. • 1: Watermark reached.

Chapter 15 Event Queue Application

15.1 Overview of Event Queue application

Application ID	0x10
Default speed	488 Hz
Configuration registers	Start on page 156.
Status registers	Start on page 158.

This application manages a queue of asynchronous events. The size of the queue is flexible and can be configured by writing to the queue_size word within the configuration registers of the Event Queue Application.

The size of the Event Queue is limited by the available RAM. The amount of RAM may be reduced by the Data FIFO Application which also can be configured to use the RAM.

Some applications inside the MMA955xL platform have an eight-bit event mask register within configuration-registers structure that selects the specific event to be stored into the Event Queue. The user can set an application's event-mask bit to enable or prevent that application's events from being added to the Event Queue.

When the selected event happens, Event Queue Application calls a routine with the following prototype to push the application's event into the Event Queue:

```
void eventQueue_push(void *data, uint8_t size)
```

With these variables:

- Data: Pointer to the first element of the data to be stored into the queue. The data must have the order: APP_ID + Payload bytes.
- Size: Amount of data (in bytes) to be stored into the queue.

Each entry into the Event Queue is of fixed size and has the following format:

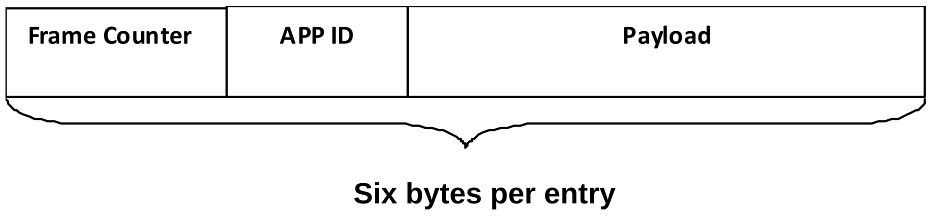


Figure 15-1. Entry format for the Event Queue

If an application has its payload less than three bytes, the Event Queue will store the data and complete the entry padding with zeros. If the application has a payload more than three bytes, the Event Queue will calculate the number of entries that this payload needs to be successfully stored, split in the format shown in the preceding illustration.

If an application tries to push data into the Event Queue and there is not enough space to store it, the Event Queue will ignore the attempt. This will be transparent for the application since there is no communication between the application and the Event Queue.

15.1.1 Modes of operation

The Event Queue works only in one mode. It stores data and stops when the end of the queue is reached and asserts an overflow flag. The queue is not circular.

The application has a configurable watermark that asserts a flag when entries have reached the configured point.

15.1.2 Reading process

To read the Event Queue, the host sends a read-status command to the MMA955xL device, calling the Event Queue application that stores the data within the queue into a buffer and returns the number of bytes read. If the number of requested bytes is not a multiple of the entry size, the pop routine ignores the extra bytes and tells the host how many effective bytes were popped.

To read data stored by the Event Queue application, the host must send a Read Data command along with a specific offset value. The following table shows the command for reading the data stored by the Event Queue Application.

Reading Event Queue example

The following example shows how to configure the Event Queue application.

Example 15-1.

1. MB0: Set APP_ID to (0x10).

   Selects the Data-FIFO application.
2. MB1: Command to (0x20).

   Sends a Write Configuration space command.
3. MB2: Set offset to zero (0x00).

   Starts writing configuration values at register 0.
4. MB3: Set count field to (0x06).

   Sends nine bytes.
5. MB4: Send Data (0x00).

   Sets Queue size.
6. MB5: Send Data (0x50).
7. MB6: Send Data (0x00).

   Watermark
8. MB7: Send Data (0x05).
9. MB8: Send Data (0x00).

   Timeout.
10. MB6: Send Data (0x00).

Bytes to Send: 0x10, 0x20, 0x00, 0x06, 0x00, 0x50, 0x00, 0x05, 0x00, 0x00.

The configuration registers can only be written once. If the Event Queue must be re-configured, the device must be reset and the configuration re-written.

Some data now must be sent to the event queue. In this example, the Low-g application is configured to send an event.

Example 15-2.

1. MB0: Set APP_ID to (0x09).
   Specifies the Low-g application.
2. MB1: Command to (0x20).
   Sends a Write Configuration space command.
3. MB2: Set offset to zero (0x09).
4. Start writing configuration values at Register 9.
5. MB3: Set count field to (0x01).
   Sends nine bytes.
6. MB4: Send Data (0x08) - Event Mask Register.
   Enables Low-g events.

Bytes to Send: 0x09, 0x20, 0x09, 0x01, 0x08.

In this example, all other Low-g configuration register values are at the reset default values.

The device powers up in Sleep mode, but it may become necessary to wake the part. That process is done in the following example.

Example 15-3.

1. MB0: Set App_ID to (0x12).
   Selects the Power Controller modes.
2. MB1: Command to (0x20).
   Sends a write configuration, with offset = 0.
3. MB2: Set offset to (0x06).
4. MB3: Set count field to (0x01).
5. MB4: Send the data value (0x00).

Bytes to Send: 0x12, 0x20, 0x06, 0x01, 0x00.

The device is enabled and configured to have an event Queue and send Low-g events to the Event Queue.

To cause a Low-g that can be measured on the X, Y, and Z axes, toss the board into the air. Read the Event Queue applications status register to get the data.

To set up for reading the status register, send the command in the following example.

Example 15-4.

1. MB0: Set APP_ID to (0x10).

   Selects the Event Queue application.
2. MB1: Command to (0x30).

   Sends a Read Status space command.
3. MB2: Set offset to zero (0x03).

   Start reading status values at register 0.
4. MB3: Set count field to (0x20).

   Reads 10 bytes (3 status + APP_ID+6 data)

Bytes to Send: 0x10, 0x30, 0x03, 0x20.

The following response is returned:

10 80 13 20 09 95 AC 09 0F 00 00 95 BE 09 0F 00 00 95 D0 09 0F 00 00

MB0: APP_ID = 0x10; Event Queue application

MB1: STATUS = 0x80; Command Complete, no errors

MB3: Actual Data Count = 0x13; 10 bytes of data

MB2: Requested Data count = 0x20; 10 bytes of data

MB4: 0x09 = Event Queue Status; Watermark and Overflow flags are set

MB5-6: = 0x95AC; Timestamp

MB7: APP ID of the application providing the actual data = 0x09; the AFE application

MB8-10: 0x0F0000 = 0x0F is the status register from the Low-g applications; 0x0F indicates a Low-g event on all axes.

MB5-6: = 0x95BE; Timestamp

MB7: APP ID of the application providing the actual data = 0x09; the AFE application

MB8-10: 0x0F0000

MB5-6: = 0x95D0; Timestamp

MB7: APP ID of the application providing the actual data = 0x09; the AFE applications

MB8-10: 0x0F0000

The offset is fixed to three bytes.

When the command is sent, the first four bytes that the MMA955xL device returns correspond to the response of the command.

The next byte is the status from the Event Queue application. This fifth byte is part of the bytes requested by the host, so it is very important that the user add one byte to the number of bytes to be read.

Due to the application's read functionality, if the application is storing data at the exact moment a read is issued by the host, the MMA955xL device will return an error condition. When this occurs, the host must retry reading the data. If any other error is returned, the host must take the proper action.

15.1.3 Event Queue block diagram

The following figure shows the Event Queue Data Flow Model that receives the Application ID and *N* size of the payload from a specific application. The Event Queue logic appends the frame counter with this data and pushes the entry (six bytes in size) into the queue. If the queue is full, the logic will ignore the push request.

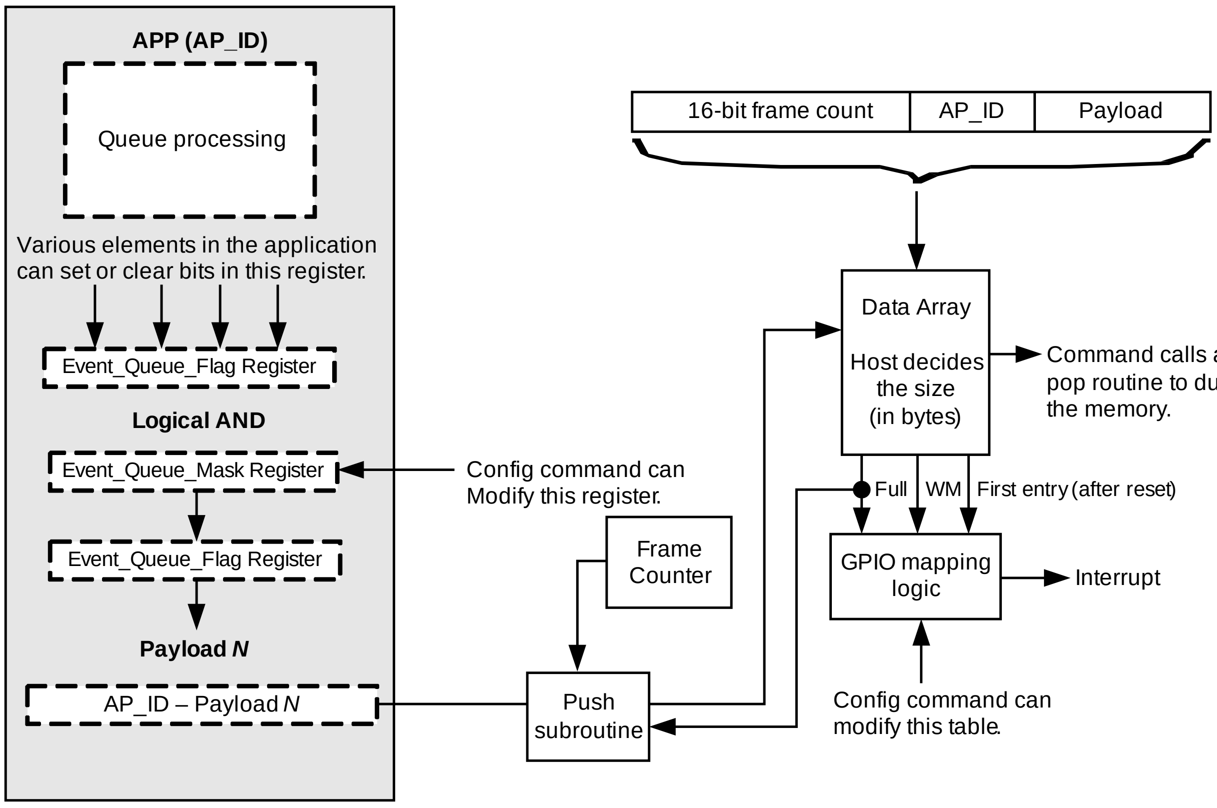

APP (AP_ID)
Queue processing
Various elements in the application can set or clear bits in this register.
Event_Queue_Flag Register
Logical AND
Event_Queue_Mask Register
Config command can Modify this register.
Event_Queue_Flag Register
Payload N
AP_ID – Payload N
16-bit frame count
AP_ID
Payload
Data Array
Host decides the size (in bytes)
Command calls a pop routine to dump the memory.
Full
WM
First entry (after reset)
Frame Counter
GPIO mapping logic
Interrupt
Push subroutine
Config command can modify this table.

Figure 15-2. Event Queue data flow

15.2 Event Queue configuration registers

The Event Queue application's configuration can only be written once. In order to re-configure the Event Queue, the whole device must be reset. This is because the Event Queue application requests RAM and RAM can only be allocated one time.

15.2.1 queue_size

Table 15-1. queue_size registers

Offset	0x00(MSB)								0x01(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	queue_size															
Reset																

Table 15-2. queue_size bit description

Field	Description
queue_size	Defines the size in bytes that the Event Queue will reserve, in order to store the data sent by any application. Units: Bytes Range of valid values: 0 - available RAM. 0x1C8 (decimal 456) is the maximum amount of RAM that can be requested from the MMA955xL device. The RAM is shared by multiple applications, so be careful not to set the FIFO to use more memory than is physically available. That will cause unknown and undesirable results.

15.2.2 queue_wmrk

Table 15-3. queue_wmrk registers

Offset	0x02(MSB)								0x03(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	queue_wmrk															
Reset																

Table 15-4. queue_wmrk bit description

Field	Description
queue_wmrk	Sets the count, in bytes, for the Event Queue to set or clear the watermark flag. Units: Bytes. Range of valid values: 0 to available bytes in the buffer.

15.2.3 queue_timeout

Table 15-5. queue_timeout registers

Offset	0x04(MSB)								0x05(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	queue_timeout															
Reset																

Table 15-6. queue_timeout bit description

Field	Description
queue_timeout	Sets the count, in cycles, for the Event Queue to set or clear the time-out flag. Units: Time; [queue_timeout] * [1/SR_{queue}]). Range of valid values: 0 to 65,535. SR_queue is the sample rate of the Event Queue application, which defaults to 488 Hz.

15.3 Event Queue status registers

15.3.1 records_number

Table 15-7. records_number registers

Offset	0x00(MSB)								0x01(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	records_number															
Reset																

Table 15-8. records_number bit description

Field	Description
records_number	Shows the current number of entries or records stored in the queue. Units: Entries (Entry = six bytes). Range of valid values: 0 to 65,535. Realistically this number should never be more than the available RAM divided by the payload size.

15.3.2 entry_size

Table 15-9. entry_size registers

Offset	0x02							
Bit	7	6	5	4	3	2	1	0
Set-bit values	0	0	0	0	0	1	1	0
Field	entry_size							
Reset	0	0	0	0	0	1	1	0

Table 15-10. entry_size bit description

Field	Description
entry_size	Shows the size in bytes of each entry or record in the queue. This value is six bytes fixed and is not user-configurable. Units: Bytes. Range of valid values: Fixed to six.

15.3.3 queue_status

Table 15-11. queue_status registers

Offset	0x03							
Bit	7	6	5	4	3	2	1	0
Field	on_going_push	on_going_pop	Reserved		to_flag	ovf_flag	empty_flag	wmrk_flag
Reset								

Table 15-12. queue_status bit descriptions

Field	Description
on_going_push	Indicates that a push operation is being executed. Units: Non-dimensional. Range of valid values: 0 to 1.
on_going_pop	Indicates that a pop operation is being executed. Units: Non-dimensional.. Range of valid values: 0 to 1.
Reserved	Bit field reserved.
to_flag	Indicates when the Event Queue has not been read for the time configured by the “queue TimeOut” parameter once the overflow marker has been reached. Units: Non-dimensional. Range of valid values: 0 to 1.
ovf_flag	Indicates when the Event Queue is full and no more entries can be stored. Units: Non-dimensional. Range of valid values: 0 to 1.
empty_flag	Indicates when the Event Queue is empty, no records or entries having been stored. Units: Non-dimensional. Range of valid values: 0 to 1.
wmrk_flag	Indicates when the numbers of bytes within the Event Queue have reached the watermark. Units: Non-dimensional. Range of valid values: 0 to 1.

Chapter 16 Status Register Application

16.1 Overview of Status Register application

Application ID	0x11
Default speed	488 Hz
Configuration registers	Start on page 162.
Status registers	Start on page 167.

The Status Register Application provides a simple way for users to combine specific status information bits from multiple applications and read that combined information from one place.

The Status Register Application configures the output of its status register by mapping a specific status register bit to a specific output bit of a specific application. This enables the Status Register Application to provide a combined status from the selected bits of user-specified applications.

There are eight, user-configurable bits in the Status Register. Each bit mirrors an output bit in the related application.

The following example maps some of the output bits of the Portrait Landscape Application (P/L) to the Status Register. The example uses the P/L application ID (0x07) and maps the following bits:

- P/L output bit 0 to Status Register bit 0
- P/L output bit 1 to Status register bit 1

  The two P/L bit are the back/front orientation bits.

Example 16-1.

1. MB0: Set the “APP_ID: Status Register Application” (0x11) application identifier.
2. MB1: Set the “Command: Write Config” (0x20).
3. MB2: Set the Offset to (0x00) to point to the start of the configuration register.
4. MB3: Set the Count field to (0x04) to indicate four bytes of data will follow.
5. MB4: Set the DATA (0x07) Portrait/Landscape application ID.
6. MB5: Set the DATA (0x00) Portrait/Landscape application status bit 0.
7. MB6: Set the DATA (0x07) Portrait/Landscape application ID.
8. MB7: Set the DATA (0x00) Portrait/Landscape application status bit 1.

Bytes to Send: 0x11, 0x20, 0x00, 0x04, 0x07, 0x00, 0x07, 0x00.

16.2 Status Register configuration registers

The following tables show the configuration registers for the Status Register Application. The bit descriptions are given in Table 16-17 on page 166.

16.2.1 APP_ID SR_00

Table 16-1. APP_ID SR_00 register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID SR_00							
Reset	0x07							

16.2.2 Output_Bit_ID SR_00

Table 16-2. Output_Bit_ID SR_00 register

Offset	0x01							
Bit	7	6	5	4	3	2	1	0
Field	Output_Bit_ID SR_00							
Reset	0x00							

16.2.3 APP_ID SR_01

Table 16-3. APP_ID SR_01 register

Offset	0x02							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID SR_01							
Reset	0x07							

16.2.4 Output_Bit_ID SR_01

Table 16-4. Output_Bit_ID SR_01 register

Offset	0x03							
Bit	7	6	5	4	3	2	1	0
Field	Output_Bit_ID SR_01							
Reset	0x01							

16.2.5 APP_ID SR_02

Table 16-5. APP_ID SR_02 register

Offset	0x04							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID SR_02							
Reset	0x07							

16.2.6 Output_Bit_ID SR_02

Table 16-6. Output_Bit_ID SR_02 register

Offset	0x05							
Bit	7	6	5	4	3	2	1	0
Field	Output_Bit_ID SR_02							
Reset	0x02							

16.2.7 APP_ID SR_03

Table 16-7. APP_ID SR_03 register

Offset	0x06							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID SR_03							
Reset	0x07							

16.2.8 Output_Bit_ID SR_03

Table 16-8. Output_Bit_ID SR_03 register

Offset	0x07							
Bit	7	6	5	4	3	2	1	0
Field	Output_Bit_ID SR_03							
Reset	0x03							

16.2.9 APP_ID SR_04

Table 16-9. APP_ID SR_04 register

Offset	0x08							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID SR_04							
Reset	0x07							

16.2.10 Output_Bit_ID SR_04

Table 16-10. Output_Bit_ID SR_04 register

Offset	0x09							
Bit	7	6	5	4	3	2	1	0
Field	Output_Bit_ID SR_04							
Reset	0x04							

16.2.11 APP_ID SR_05

Table 16-11. APP_ID SR_05 register

Offset	0x0A							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID SR_05							
Reset	0x00							

16.2.12 Output_Bit_ID SR_05

Table 16-12. Output_Bit_ID SR_05 register

Offset	0x0B							
Bit	7	6	5	4	3	2	1	0
Field	Output_Bit_ID SR_05							
Reset	0x00							

16.2.13 APP_ID SR_06

Table 16-13. APP_ID SR_06 register

Offset	0x0C							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID SR_06							
Reset	0x07							

16.2.14 Output_Bit_ID SR_06

Table 16-14. Output_Bit_ID SR_06 register

Offset	0x0D							
Bit	7	6	5	4	3	2	1	0
Field	Output_Bit_ID SR_06							
Reset	0x06							

16.2.15 APP_ID SR_07

Table 16-15. APP_ID SR_07 register

Offset	0x0E							
Bit	7	6	5	4	3	2	1	0
Field	APP_ID SR_07							
Reset	0x00							

16.2.16 Output_Bit_ID SR_07

Table 16-16. Output_Bit_ID SR_07 register

Offset	0x0F							
Bit	7	6	5	4	3	2	1	0
Field	Output_Bit_ID SR_07							
Reset	0x00							

Table 16-17. Status Register bit descriptions

Field	Description
APP_ID (APP_ID SR_n) [7:0]	The application identifier. Zero value and 0xFF are reserved. Units: None. Range of valid values: [0 to 31] and 0xFF, realistically up to 0x1F.
Bit_ID (Output_Bit_ID SR_n) [7:0]	The bit number to be mapped on the Status Register bit *n*. Units: Non-dimensional. Range of valid values: [0 to 255].

COMMENT: 32 data registers x8 bits each = 256 possibilities (selection is similar to as GPIO_Application, give an practical example as well...) probably similar to the GPIO application.

16.3 Status Register default configuration

After reset, the *status register* configuration registers contains the Portrait LAPO, BAFRO, and Z-Tilt lockout bits as well as the Data Ready and Command Complete bits.

NOTE

The upper two bits in the upper byte are fixed, but the lower eight bits in the lower byte can be remapped by the user to any application and bits.

Table 16-18. Status Register MSB

Offset	0x00							
Bit	15	14	13	12	11	10	9	8
Field	Command Complete	Data Ready	N/A	N/A	N/A	N/A	N/A	N/A
Reset								

Table 16-19. Status Register LSB

Offset	0x01							
Bit	7	6	5	4	3	2	1	0
Field	Unassigned	PL Z-Tilt Lockout	Unassigned	PL LAPO			PL BAFRO	
Reset								

The Status Register Application's default settings are configured to meet the needs of most users. (See the following table.)

The Portrait/Landscape Application output bits are mapped to the status bits. For detailed information about the Portrait/Landscape Application's outputs, see Table 9-16 on page 102.

Table 16-20. Status Register application's default settings

Field	Description
PL Z-Tilt Lockout	Reports a Z-tilt angle lockout. Range of valid values: • 0: Lockout condition has not been detected. • 1: Z-tilt lockout trip angle has been exceeded. Lockout has been detected.
PL LAPO	Reports a new portrait/landscape orientation event. Range of valid values: • 000: Undefined. The default, power-up state. • 001: Portrait up – Device is standing vertically in the normal orientation. • 010: Portrait down – Device is standing vertically in the inverted orientation. • 011: Landscape right – Device is in the landscape mode to the right. • 100: Landscape left – Device is in the landscape mode to the left.
PL BAFRO	Reports a new back or front orientation event. Range of valid values: • 00: Undefined – This is the default power-up state. • 01: Front – Device is in the front-facing orientation. • 11: Back – Device is in the back-facing orientation.

Chapter 17 Sleep/Wake Application

17.1 Overview of Sleep/Wake application

Application ID	0x12
Default speed	488 Hz
Configuration registers	Start on page 172.
Status registers	Start on page 176.

This application configures and controls the power-control modes of the accelerometer provides configuration flexibility for minimizing power consumption.

The application has three modes of operation: Run, Doze, and Sleep. The Sleep/Wake module puts the accelerometer into Doze mode automatically when no movement is detected. When a change in orientation or movement above the threshold is detected for the specified time period, the application returns to the Run mode.

To save a significant amount of power, only run the calculation-intensive applications when the accelerometer is in motion.

Using the activity level settings, some tasks may easily be bypassed when the accelerometer is sleeping. For example, it may not be necessary to run the Portrait / Landscape application while the device is sitting undisturbed flat on a desk top.

17.1.1 Run mode

In Run mode, all applications are scheduled to run at their maximum established frame rate. An application enters Run mode if the following conditions are met:

- The GPIO interrupt is asserted (RGPIO4/INT)
- A write command is issued from the host to the accelerometer
- Movement above the threshold is detected and the previous state was Doze mode

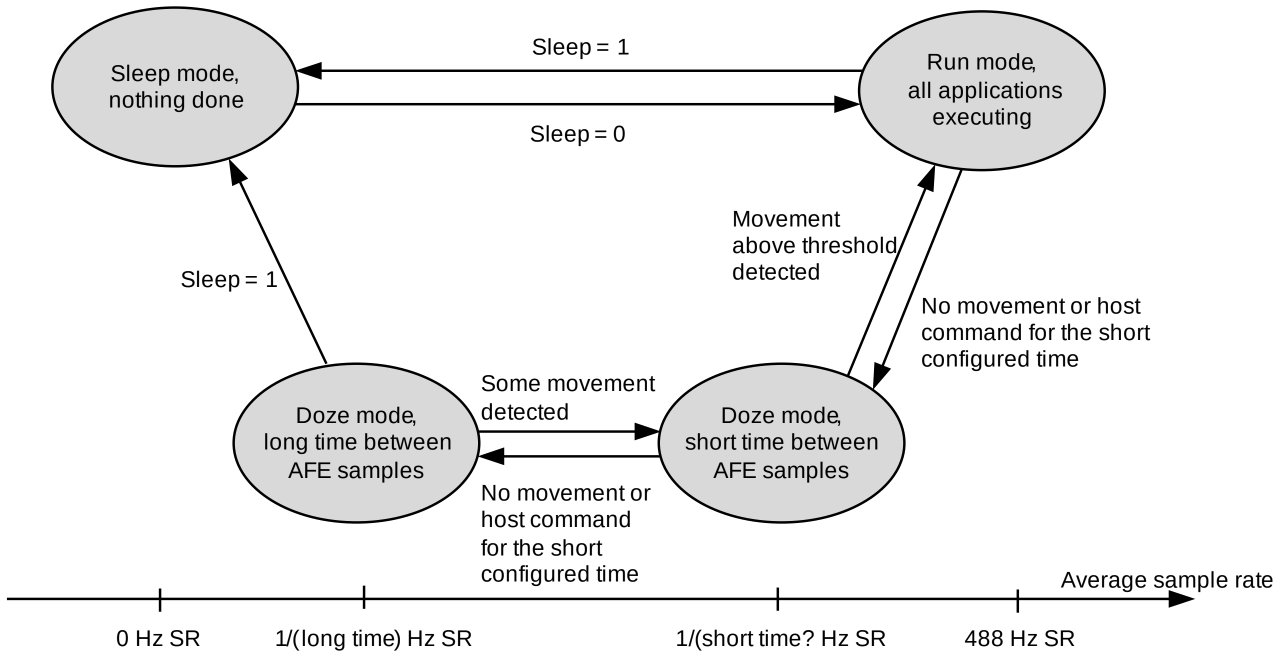

Sleep mode, nothing done
Run mode, all applications executing
Sleep = 1
Sleep = 0
Sleep = 1
Movement above threshold detected
No movement or host command for the short configured time
Doze mode, long time between AFE samples
Some movement detected
Doze mode, short time between AFE samples
No movement or host command for the short configured time
Average sample rate
0 Hz SR
1/(long time) Hz SR
1/(short time? Hz SR
488 Hz SR

Figure 17-1. Platform's power modes state diagram

17.1.2 Doze mode

In Doze mode, the application only executes four AFE samples at a sample rate defined by the user. The user can configure two sample rate values: long time and short time. These parameters are configured in the **long_time_off** and **short_time_off** registers, respectively.

The short-time sample rate is used when the sensor detects some movement, but not enough to change to Run mode.

The long-time sample rate is used when the Sleep/Wake module detects no movement in the accelerometer.

The application enters Doze mode when the Sleep/Wake module detects no movement in the accelerometer for a specified period of time. The amount of time is configured in the **doze_thresh** register.

17.1.3 Sleep mode

In Sleep mode, the MMA955xL device does nothing and remains in the lowest-power mode. The device can enter this mode only when the user sets the SNCEN bit from the **cfg** configuration register or when the application starts. Since the SNCEN bit is set by default, the application starts in Sleep mode.

To exit the Sleep mode, the user must clear the SNCEN bit from the **cfg** configuration register. This is done by issuing the corresponding write configuration command through the slave communications interface.

If the sensor is being used as a standalone, without a slave communication interface, the user must overwrite the default configuration to have the sensor start in the Run mode. To overwrite the default configuration, a user application must be added. Since each task must have an initialization function, the user must add the write instruction of the **cfg** register to change the sensor from Sleep to Run mode.

This configuration can only be done in the initialization function or the sensor will start in Sleep mode and never run the user applications. The following example shows how to overwrite the default configuration of the **cfg** register.

Example 17-1.

```
void initCbFunction(void)
{
   /* power control structure pointer */
   power_data_struct_t *sleep_ptr;

   /* Assign power control structure address to pointer */
   sleep_ptr = GetDataPtr(LONG_SHORT_INT_APP_ID);

   /* configure power ctrl module to be in run mode */
   sleep_ptr->param.pc_cfg = 0x00;

   ...
}
```

17.2 Sleep/Wake configuration registers

The following sections give the configuration registers for the Sleep/Wake application.

17.2.1 sensitivity_thresh

Table 17-1. sensitivity_thresh registers

Offset	0x00(MSB)								0x01(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	sensitivity_thresh								sensitivity_thresh							
Reset	0x00								1	0	1	0	0	0	0	0

Table 17-2. sensitivity_thresh bit description

Field	Description
sensitivity_thresh	Configures the movement threshold of the application to change from Doze to Run mode. Units: Non-dimensional. Range of valid values: 120 to 300.

17.2.2 doze_time_thresh

Table 17-3. doze_time_thresh registers

Offset	0x02(MSB)								0x03(LSB)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	doze_thresh								doze_thresh							
Reset	0	0	0	0	1	1	1	1	1	0	1	0	0	0	0	0

Table 17-4. doze_time_thresh bit description

Field	Description
doze_thresh	Configures the time that the device must be still before entering Doze mode. For example, if the system sample rate if 488 Hz, then to set a time of 1 second, write 488 (0x1EB) to this register. Units: Algorithm cycles (Time = [Algorithm cycles] * [1/$SR_{PWRCTRL}$]). Range of valid values: 1 to 65,535

17.2.3 long_time_off

Table 17-5. long_time_off registers

Offset	0x04							
Bit	7	6	5	4	3	2	1	0
Field	Reserved				long_time_off [3:0]			
Reset	0x00				0	1	1	1

Table 17-6. long_time_off bit description

Field	Description
Reserved	Bit field reserved.
long_time_off	Configures the long-time interval between AFE samples when the application is in Doze mode. Units: Time. (For more information, see Section 10.3, “High-g/Low-g status register,” on page 115) Range of valid values: from 0 to 10. Long-time valuesTime (seconds) • 0. . . . . 4.1 ms • 1. . . . . 8.19 ms • 2. . . . . 16.4 ms • 3. . . . . .32 ms • 4. . . . . 65.5 ms • 5. . . . . 131 ms • 6. . . . . 162 ms • 7. . . . . 524 ms • 8. 1.05s • 9. 2.1s • 10. . . . . 4.19s

17.2.4 short_time_off

Table 17-7. short_time_off register

Offset	0x05							
Bit	7	6	5	4	3	2	1	0
Field	Reserved				short_time_off [3:0]			
Reset	0x00				0	1	1	0

Table 17-8. short_time_off bit description

Field	Description
Reserved	Bit field reserved.
short_time_off	Configures the short-time interval between AFE samples when the application is in Doze mode. Units: Time. (For more information, see Section 10.3, “High-g/Low-g status register,” on page 115) Range of valid values: 0 to 10. Long-time valuesTime (seconds) • 0. . . . . 4.1 ms • 1. . . . . 8.19 ms • 2. . . . . 16.4 ms • 3. . . . . .32 ms • 4. . . . . 65.5 ms • 5. . . . . 131 ms • 6. . . . . 162 ms • 7. . . . . 524 ms • 8. 1.05s • 9. 2.1s • 10. . . . . 4.19s

17.2.5 cfg

Table 17-9. cfg register

Offset	0x06							
Bit	7	6	5	4	3	2	1	0
Field	Reserved			Stop_DIS	IRQ_EN	SCHEN	FLEEN	SNCEN
Reset	0x00			0	0	0	0	1

Table 17-10. cfg bit descriptions

Field	Description
Reserved	Bit field reserved.
Stop_DIS	Disables or enables the low-power mode. If this bit is set, the device doesn't execute the STOP assembler instruction even if the SNCEN, FLEEN, or SCHEN bits are set as 1b. • 0: Enables STOP mode. • 1: Disables STOP mode.
IRQ_EN	Enables or disables the IRQ interruption in the GPIO 4. • 0: Disables IRQ interruption. • 1: Enables IRQ interruption.
SCHEN	Enables or disables the Doze mode. • 0: Disables the Doze mode. • 1: Enables the Doze mode.
FLEEN	Controls the use of long- and short-time between AFE samples when in Doze mode. If this bit is cleared, the application does not use the long- and short-time values to determine the time between AFE samples. Instead, the time is fixed as 2.05 ms. • 0: Time between AFE samples fixed at 2.05 ms. • 1: Long- and short-time values determine the time between AFE samples.
SNCEN	Enables or disables the Sleep mode. **Note:** When this bit is set, the device will enter Sleep mode. The device can only exit this mode is by clearing this bit via a configuration-write command. • 0: Disables the Sleep mode. • 1: Enables the Sleep mode.

17.3 Sleep/Wake status registers

17.3.1 scheduler_mode

Table 17-11. scheduler_mode register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	—	—	—	—	—	—	scheduler_mode	
Reset	0	0	0	0	0	0	0	0

Table 17-12. scheduler_mode bit description

Field	**Description**
scheduler_mode	Shows the scheduler mode when the SCHED and FLEEN bits are set in the cfg register. If those bits are not set, the value of this register may not reflect the actual scheduler mode. Units: None. Range of valid values: 0 to 0x02. • 00: Doze mode. Frame rate determined by the **long_time_off** configuration register. • 01: Doze mode. Frame rate determined by the **short_time_off** configuration register. • 10: Run mode.

Chapter 18 Reset/Suspend/Clear Control Application

18.1 Overview of Reset/Suspend/Clear Control application

Application ID	0x17
Default speed	488 Hz
Configuration registers	Start on page 178 and page 184.
Status registers	None.

This application provides a way to reset, suspend, and clear the outputs of the applications in the MMA955xL device. The reset and clear functions are implemented in each application as call-back functions. The suspend function is handled in the Scheduler Application.

One of the requirements of an application on the MMA955xL device is that it have a reset and clear function that can be called by the scheduler or another application. This chapter describes how an application's reset and clear callback functions can be called or triggered.

There are three groups of configuration registers for the Reset/Suspend/Clear Control application.

18.1.1 Reset

The reset bit, when set, schedules a reset for an application. At the next system cycle, the reset process is handled. The reset bit is automatically cleared by the Scheduler Application.

When the reset bit for an application is set, the following actions occur:

- The application's reset callback function is executed.

  The reset function is part of an application and it typically resets the application's outputs and internal variables.
- The reset bit is cleared.
- The scheduled application is executed.

18.1.2 Suspend

The suspend bit, when set, prevents the an application from executing while the suspend bit is set. Setting or clearing the suspend flag is managed by the host through a command or by an application by the API functions. (For more information on the latter, see "API functions" on page 195.)

To preserve data coherency in an application, the suspend flag must be set before attempting to modify an application's configuration. The bit then must be cleared after the configuration has been changed.

An application's reset callback function is executed when its suspend bit flag is set, so such callbacks should be implemented in a way to manage reset and suspend conditions.

18.1.3 Clear

The clear bit clears an application's outputs. The clear flag is automatically cleared by the scheduler.

When an application's clear bit is set, it causes the following actions:

- The application's clear callback is executed.
  This typically resets the application's outputs.
- The clear flag is cleared.
- The application is executed.

18.2 Configuration registers for Reset/Suspend/Clear Control applications

18.2.1 Reset configuration register

Table 18-1. Reset registers

Offset	(MSB) 0x00 = reset_bits[31:24]							
Bit	31	30	29	28	27	26	25	24
Field	Reserved		User Application	User Application	User Application	User Application	User Application	User Application
Reset	1	1	1	1	1	1	1	1
Offset	0x01 = reset_bits[23:16]							
Bit	23	22	21	20	19	18	17	16
Field	Rst/Susp/Clr	User Application	User Application	User Application	User Application	Auto-Wake/ Sleep	Status	Event FIFO
Reset	0	1	1	1	1	1	1	1
Offset	0x02 = reset_bits[15:8]							
Bit	15	14	13	12	11	10	9	8
Field	Data FIFO	Frame Counter	User Application	User Application	Tilt Sensing	Tap Detect	LG	HG
Reset	1	1	1	1	1	1	1	1
Offset	(LSB) 0x03 = reset_bits[7:0]							
Bit	7	6	5	4	3	2	1	0
Field	PL	AFE	Reserved	MBOX	GPIO	CI	Scheduler	General Rst
Reset	1	1	1	1	1	1	1	1

Table 18-2. Reset bit descriptions

Field	Description
Reserved [31:30]	Not used.
User application [29:24]	These are bits to use to reset user applications assigned to the following APP_IDs 0x1D, 0x1C, 0x1B, 0x1A, 0x19, and 0x18. • 0: Normal Operation • 1: Initiates the reset sequence if the user application
Reset/Suspend/Clear [23]	• 0: Normal operation. • 1: Initiates the reset sequence of the Reset/Suspend/Clear application.
User application [22:19]	These are bits to use to reset user applications assigned to APP_IDs 0x16, 0x15, 0x14, and 0x13. • 0: Normal Operation • 1: Initiates the reset sequence if the user application
Auto-Wake/Sleep [18]	• 0: Disables the reset sequence of the Auto-Wake/Sleep application. • 1: Initiates the reset sequence of the Auto-Wake/Sleep application.
Reserved [17]	Not used.
Event-FIFO [16]	• 0: Disables the reset sequence of the Event-Queue application. • 1: Initiates the reset sequence of the Event-Queue application.
Data FIFO [15]	• 0: Normal operation. • 1: Initiates the reset sequence of the Data FIFO application.
Frame Counter [14]	• 0: Disables the reset sequence of the Frame Counter application. • 1: Initiates the reset sequence of the Frame Counter application.
User Application [13:12]	These are bits to use to reset user applications assigned to APP_IDs 0x0D and 0x0C. • 0: Normal operation. • 1: Initiates the reset sequence if the user application
Tilt Sensing [11]	• 0: Disables the reset sequence of the Tilt Sensing application. • 1: Initiates the reset sequence of the Tilt Sensing application.
Tap Detection [10]	• 0: Disables the reset sequence of the Tap Detection application. • 1: Initiates the reset sequence of the Tap Detection application.
Low-g Detection [9]	• 0: Disables the reset sequence of the Low-g detection application. • 1: Initiates the reset sequence of the Low-g detection application.
High-g Detection [8]	• 0: Disables the reset sequence of the High-g detection application. • 1: Initiates the reset sequence of the High-g detection application.
Portrait/Landscape [7]	• 0: Disables the reset sequence of the portrait/landscape application. • 1: Initiates the reset sequence of the portrait/landscape application.
AFE [6]	• 0: Disables the reset sequence of the front-end application. • 1: Initiate the reset sequence of the front-end application.
Reserved [5]	Reserved.
Mailbox [4]	• 0: Disables the reset sequence of the mailbox application. • 1: Initiates the reset sequence of the mailbox application.

Table 18-2. Reset bit descriptions (continued)

Field	Description
GPIO [3]	• 0: Disables the reset sequence of the GPIO application. • 1: Initiates the reset sequence of the GPIO application.
Command Interpreter [2]	• 0: Disables the reset sequence of the command-interpreter application. • 1: Initiates the reset sequence of the command-interpreter application.
Scheduler [1]	• 0: Disables the reset sequence of the scheduler-application. • 1: Initiates the reset sequence of the scheduler-application.
General Reset [0]	• 0: Disables a system-wide reset sequence of all MMA955xLapplications. • 1: Initiates a system-wide reset sequence of all MMA955xLapplications.

18.2.2 Suspend configuration register

Table 18-3. Suspend registers

Offset	(MSB) 0x04 = reset_bits[31:24]							
Bit	31	30	29	28	27	26	25	24
Field	Reserved		User Application	User Application	User Application	User Application	User Application	User Application
Reset	1	1	1	1	1	1	1	1
Offset	0x05 = reset_bits[23:16]							
Bit	23	22	21	20	19	18	17	16
Field	Rst/Susp/Clr	User Application	User Application	User Application	User Application	Auto-Wake/ Sleep	Status	Event FIFO
Reset	0	1	1	1	1	1	1	1
Offset	0x06 = reset_bits[15:8]							
Bit	15	14	13	12	11	10	9	8
Field	Data FIFO	Frame Counter	User Application	User Application	Tilt Sensing	Tap Detect	LG	HG
Reset	1	1	1	1	1	1	1	1
Offset	(LSB) 0x07 = reset_bits[7:0]							
Bit	7	6	5	4	3	2	1	0
Field	PL	AFE	Reserved	MBOX	GPIO	CI	Scheduler	General Rst
Reset	1	1	1	1	1	1	1	1

Table 18-4. Suspend bit descriptions

Field	Description
Reserved [31:30]	Not used.
User application [29:24]	These are bits to use to reset user applications assigned to the following APP_IDs 0x1D, 0x1C, 0x1B, 0x1A, 0x19, and 0x18. • 0: Normal Operation • 1: Initiates the reset sequence if the user application
Reset/Suspend/Clear [23]	• 0: Normal operation. • 1: Initiates the reset sequence of the Reset/Suspend/Clear application.
User application [22:19]	These are bits to use to reset user applications assigned to APP_IDs 0x16, 0x15, 0x14, and 0x13. • 0: Normal Operation • 1: Initiates the reset sequence if the user application
Auto-Wake/Sleep [18]	• 0: Disables the reset sequence of the Auto-Wake/Sleep application. • 1: Initiates the reset sequence of the Auto-Wake/Sleep application.
Reserved [17]	Not used.
Event-FIFO [16]	• 0: Disables the reset sequence of the Event-Queue application. • 1: Initiates the reset sequence of the Event-Queue application.
Data FIFO [15]	• 0: Normal operation. • 1: Initiates the reset sequence of the Data FIFO application.
Frame Counter [14]	• 0: Disables the reset sequence of the Frame Counter application. • 1: Initiates the reset sequence of the Frame Counter application.
User Application [13:12]	These are bits to use to reset user applications assigned to APP_IDs 0x0D and 0x0C. • 0: Normal operation. • 1: Initiates the reset sequence if the user application
Tilt Sensing [11]	• 0: Disables the reset sequence of the Tilt Sensing application. • 1: Initiates the reset sequence of the Tilt Sensing application.
Tap Detection [10]	• 0: Disables the reset sequence of the Tap Detection application. • 1: Initiates the reset sequence of the Tap Detection application.
Low-g Detection [9]	• 0: Disables the reset sequence of the Low-g detection application. • 1: Initiates the reset sequence of the Low-g detection application.
High-g Detection [8]	• 0: Disables the reset sequence of the High-g detection application. • 1: Initiates the reset sequence of the High-g detection application.
Portrait/Landscape [7]	• 0: Disables the reset sequence of the portrait/landscape application. • 1: Initiates the reset sequence of the portrait/landscape application.
AFE [6]	• 0: Disables the reset sequence of the front-end application. • 1: Initiate the reset sequence of the front-end application.
Reserved [5]	Reserved.
Mailbox [4]	• 0: Disables the reset sequence of the mailbox application. • 1: Initiates the reset sequence of the mailbox application.

Table 18-4. Suspend bit descriptions (continued)

Field	Description
GPIO [3]	• 0: Disables the reset sequence of the GPIO application. • 1: Initiates the reset sequence of the GPIO application.
Command Interpreter [2]	• 0: Disables the reset sequence of the command-interpreter application. • 1: Initiates the reset sequence of the command-interpreter application.
Scheduler [1]	• 0: Disables the reset sequence of the scheduler-application. • 1: Initiates the reset sequence of the scheduler-application.
General Reset [0]	• 0: Disables a system-wide reset sequence of all MMA955xLapplications. • 1: Initiates a system-wide reset sequence of all MMA955xLapplications.

18.3 Clear configuration register

Table 18-5. Clear registers

Offset	(MSB) 0x08 = reset_bits[31:24]							
Bit	31	30	29	28	27	26	25	24
Field	Reserved		User Application	User Application	User Application	User Application	User Application	User Application
Reset	1	1	1	1	1	1	1	1
Offset	0x09 = reset_bits[23:16]							
Bit	23	22	21	20	19	18	17	16
Field	Rst/Susp/Clr	User Application	User Application	User Application	User Application	Auto-Wake/ Sleep	Status	Event FIFO
Reset	0	1	1	1	1	1	1	1
Offset	0x10 = reset_bits[15:8]							
Bit	15	14	13	12	11	10	9	8
Field	Data FIFO	Frame Counter	User Application	User Application	Tilt Sensing	Tap Detect	LG	HG
Reset	1	1	1	1	1	1	1	1
Offset	(LSB) 0x11 = reset_bits[7:0]							
Bit	7	6	5	4	3	2	1	0
Field	PL	Frontend	Reserved	MBOX	GPIO	CI	Scheduler	General Susp
Reset	1	1	1	1	1	1	1	1

Table 18-6. Clear bit descriptions

Field	Description
Reserved [31:30]	Not used.
User application [29:24]	These are bits to use to reset user applications assigned to the following APP_IDs 0x1D, 0x1C, 0x1B, 0x1A, 0x19, and 0x18. • 0: Normal Operation • 1: Initiates the reset sequence if the user application
Reset/Suspend/Clear [23]	• 0: Normal operation. • 1: Initiates the reset sequence of the Reset/Suspend/Clear application.
User application [22:19]	These are bits to use to reset user applications assigned to APP_IDs 0x16, 0x15, 0x14, and 0x13. • 0: Normal Operation • 1: Initiates the reset sequence if the user application
Auto-Wake/Sleep [18]	• 0: Disables the reset sequence of the Auto-Wake/Sleep application. • 1: Initiates the reset sequence of the Auto-Wake/Sleep application.
Reserved [17]	Not used.
Event-FIFO [16]	• 0: Disables the reset sequence of the Event-Queue application. • 1: Initiates the reset sequence of the Event-Queue application.
Data FIFO [15]	• 0: Normal operation. • 1: Initiates the reset sequence of the Data FIFO application.
Frame Counter [14]	• 0: Disables the reset sequence of the Frame Counter application. • 1: Initiates the reset sequence of the Frame Counter application.
User Application [13:12]	These are bits to use to reset user applications assigned to APP_IDs 0x0D and 0x0C. • 0: Normal operation. • 1: Initiates the reset sequence if the user application
Tilt Sensing [11]	• 0: Disables the reset sequence of the Tilt Sensing application. • 1: Initiates the reset sequence of the Tilt Sensing application.
Tap Detection [10]	• 0: Disables the reset sequence of the Tap Detection application. • 1: Initiates the reset sequence of the Tap Detection application.
Low-g Detection [9]	• 0: Disables the reset sequence of the Low-g detection application. • 1: Initiates the reset sequence of the Low-g detection application.
High-g Detection [8]	• 0: Disables the reset sequence of the High-g detection application. • 1: Initiates the reset sequence of the High-g detection application.
Portrait/Landscape [7]	• 0: Disables the reset sequence of the portrait/landscape application. • 1: Initiates the reset sequence of the portrait/landscape application.
AFE [6]	• 0: Disables the reset sequence of the front-end application. • 1: Initiate the reset sequence of the front-end application.
Reserved [5]	Reserved.
Mailbox [4]	• 0: Disables the reset sequence of the mailbox application. • 1: Initiates the reset sequence of the mailbox application.

Table 18-6. Clear bit descriptions (continued)

Field	Description
GPIO [3]	• 0: Disables the reset sequence of the GPIO application. • 1: Initiates the reset sequence of the GPIO application.
Command Interpreter [2]	• 0: Disables the reset sequence of the command-interpreter application. • 1: Initiates the reset sequence of the command-interpreter application.
Scheduler [1]	• 0: Disables the reset sequence of the scheduler-application. • 1: Initiates the reset sequence of the scheduler-application.
General Reset [0]	• 0: Disables a system-wide reset sequence of all MMA955xLapplications. • 1: Initiates a system-wide reset sequence of all MMA955xLapplications.

Once an application has set a flag, it can be reset by one of the following mechanisms:

- Execute a power-on reset.
- Wake the device after a low-power mode.
- Direct the host to write to general, reset/suspend/clear configuration bit.
- Direct the host to write to an application reset/suspend/clear configuration bit.

18.4 Reset/Suspend/Clear status registers

There are no status registers.

18.5 Reboot to ROM CI from flash code

In order to reboot to ROM Command Interpreter, it is necessary to execute the reset callback function of the reset/suspend/clear application by setting the respective reset flag. The complete command for this operation is:

Example 18-1.

```
MBOX0 = 0x17 /*Application ID*/
MBOX1 = 0x20 /*CONFIG_W command*/
MBOX2 = 0x01 /*Offset*/
MBOX3 = 0x01 /*Number of bytes to write*/
MBOX4 = 0x80 /*Data*/
```

18.6 Reboot to flash code from ROM CI

The operation to reboot into flash code when the ROM Command Interpreter is running can be performed by sending a CI_RESET command, the mailbox settings for this command is:

Example 18-2.

```
MBOX0 = 0x29 /*ROM Command for boot to flash*/
MBOX1 = 0x00 /*Reserved*/
MBOX2 = 0xFF /*CI_PWR*/
MBOX3 = 0xFF /*CI_PWR*/
MBOX4 = 0xFF /*CI_PWR*/
MBOX5 = 0xFF /*CI_PWR*/
```

For details about the ROM CI commands, see the *MMA955xL Intelligent Motion-Sensing Platform Hardware Reference Manual* (MMA955xLRM). (See “References” on page 13.)

Chapter 19 MBOX Configuration Application

19.1 Overview of MBOX Configuration application

Application ID	0x18
Default speed	Always available
Configuration registers	page 191
Status registers	None

The MBOX Configuration application works with the Mailbox application (Chapter 7, “Mailbox Application” on page 67) and the Communications application (Chapter 5, “Communication Interface” on page 55) to provide data back to the host in a way that the host can best use the data.

The MBOX Configuration application configures how the mailboxes behave. The Mailbox application configures what data is stored in the mailboxes.

By default, the MMA955xL operates in the Commands/Response mode, where a host must issue a write command followed by one or more reads to get data. The MMA955xL can also be put into a Legacy mode, where the host just issues a read command to get data.

The mailboxes can be accessed in either of two modes: Normal or Legacy. The mailboxes’ default mode is Normal, but the mode can be changed through this the mailbox configuration application (APP_ID = 0x18).

The following figure shows the differences between the Normal and Legacy modes.

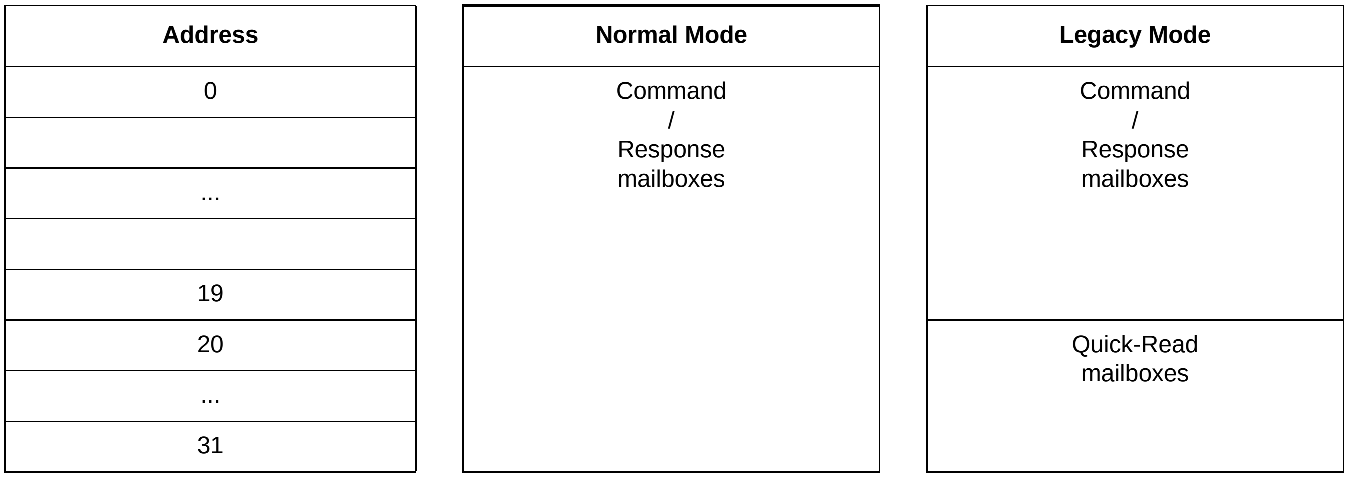


Figure 19-1. Difference between Normal and Legacy modes

19.2 Normal mode

In Normal mode, only the command/response communications model is supported. In order to read valid data from the MMA955xL platform, the host must send a command through the mailboxes and wait for the command to be processed. Then, the host must read back the mailboxes that now have the data. The host can wait for the command to be processed by polling the Command Complete (COCO) bit or the host can wait for the INT_O interrupt.

The Normal mode fully supports streaming-read transactions when the response to a command may be more than 32 bytes long.

The following procedure gives the sequence for setting up the interrupt pin to go active after a COCO:

1. MB0: Set the “APP_ID: Communication application” (0x18) application identifier.
2. MB1: Set the “Command: Write Config” (0x20) application identifier.
3. MB2: Set the Offset to Zero field (0x00) to point to the configuration register.
4. MB3: Set the Count field to (0x01).

   This is done because only one data byte needs to be sent.
5. MB4: Set the DATA (0x80) bit 7.

   This enables the interrupt pin.

Bytes to Send: 0x18, 0x20, 0x00, 0x01, 0x80.

19.3 Legacy mode

In Legacy mode, the lower-address mailboxes operate as described in the Normal mode—in the command/response communications model. MB20 through MB31, however, are used as Quick-Read registers. These registers are automatically updated at the end of each sample frame with the latest results from the chosen applications.

The Mailbox application determines what data will appear in the mailboxes.

The quick-read output data is selected with the configuration of the Mailbox application. (For more detail, see “Mailbox Application” on page 67.)

Quick-Read registers enable the host to quickly and directly read a limited set of data directly from the MMA955xL platform without first having to issue a command and wait for the completion of the command processing. This makes support easier for legacy systems that expect to read sensor data. (In Legacy mode, MB20 to MB31 are reserved for mapping the Quick-Read registers.)

By default, the Quick-Read registers (MB20-MB31) contain the following data:

MB20-21 = Status 0,1

MB22 = Event Queue status

MB23 = FIFO status

MV24-25 = AFE Frame Counter

MB26-27 = AFE, Stage 0; X-axis data

MB28-29 = AFE, Stage 0; Y-axis data

MB30-31 = AFE, Stage 0; Z-axis data

This data—in the Quick-Read registers in Legacy mode—can be changed with the Mailbox application.

19.4 Configuring mailbox operational mode

The operational mode of the slave communications interface mailbox is configured via the Mailbox Configuration application. This application allows the host system to configure the slave communications interface functions including the mailbox transaction interrupt pin mode, mailbox Normal/Legacy modes, and transaction-streaming modes.

The following example shows how to configure the mailbox operating mode from Normal to Legacy

Example 19-1.

1. MB0: Set the “APP_ID: Mailbox Mode Config” application identifier (0x18).
2. MB1: Set the “Command: Write Config” application identifier (0x20).
3. MB2: Set the Offset to Zero field (0x00) to point to the configuration register.
4. MB3: Set the Count field to 0x01 because only one data byte needs to be sent.
5. MB4: Set the DATA value to 0x10.

   This sets the Legacy field to 1b which selects the Legacy mode.

Bytes to send: 0x18, 0x20, 0x00, 0x01, 0x10.

19.5 MBOX Configuration memory map and register

19.5.1 MBOX Configuration memory map

Table 19-1. memory map

Offset address	Register	Access	Reset	Details
0x00	MBOX Configuration register	Read/Write	0x00	page 192

19.5.2 MBOX Configuration register

Table 19-2. MBOX Configuration register

Offset	0x00							
Bit	7	6	5	4	3	2	1	0
Field	INT_O_EN	INT_O_POL	INT_O_ FRAME_EN	LEGACY	UPDMODES		—	—
Reset	0	0	0	0	00		0	0

Table 19-3. Communications configuration register field descriptions

Bit(s)	Field	Description
7	INT_O_EN	Enables or disables the assertion of the INT_O signal every time a mailbox command is been processed. Range of valid values: 0 Disables the assertion of the INT_O signal. 1 Enables the assertion of the INT_O signal.
6	INT_O_POL	Configures the polarity of the INT_O signal when it is asserted. Range of valid values: 0 Active high. 1 Active low.
5	INT_O_FRAME_EN	If enabled, generates the INT_O interrupt on completion of the AFE sample—once each frame. If not enabled, the INT_O signal is generated on a command-complete basis. This bit was added to support synchronization between the host and the MMA955xL device. Data will be ready on a frame basis (488 Hz, 244 Hz, or 122 Hz). Range of valid values: 0 Interrupt not generated. 1 Interrupt generated on completion of AFE sample.
4	LEGACY	Selects between Normal and Legacy mode. Range of valid values: 0 Normal mode. 1 Legacy mode.
3–2	UPDMODES	When in Legacy mode, configures how and when the Quick-Read registers are updated. This field is valid only if the Mailbox is operating in Legacy mode. See Bit 4. Range of valid values: 00 Mode 0: Updates the Quick-Read registers (QR) whether the slave port is active or inactive. 01 Mode 1: Updates the QR registers only if the slave port is inactive. If the slave port is active, the update takes no action and waits until the slave port is inactive. 10 Mode 2: Updates the QR registers if the slave port is inactive. If the slave port is active (I^2C transactions are running), this mode will enable a receive interrupt in the slave port to be triggered immediately after the transaction ends. After this, the QR register is immediately updated and the receive interrupt disabled.
1–0	—	Reserved.

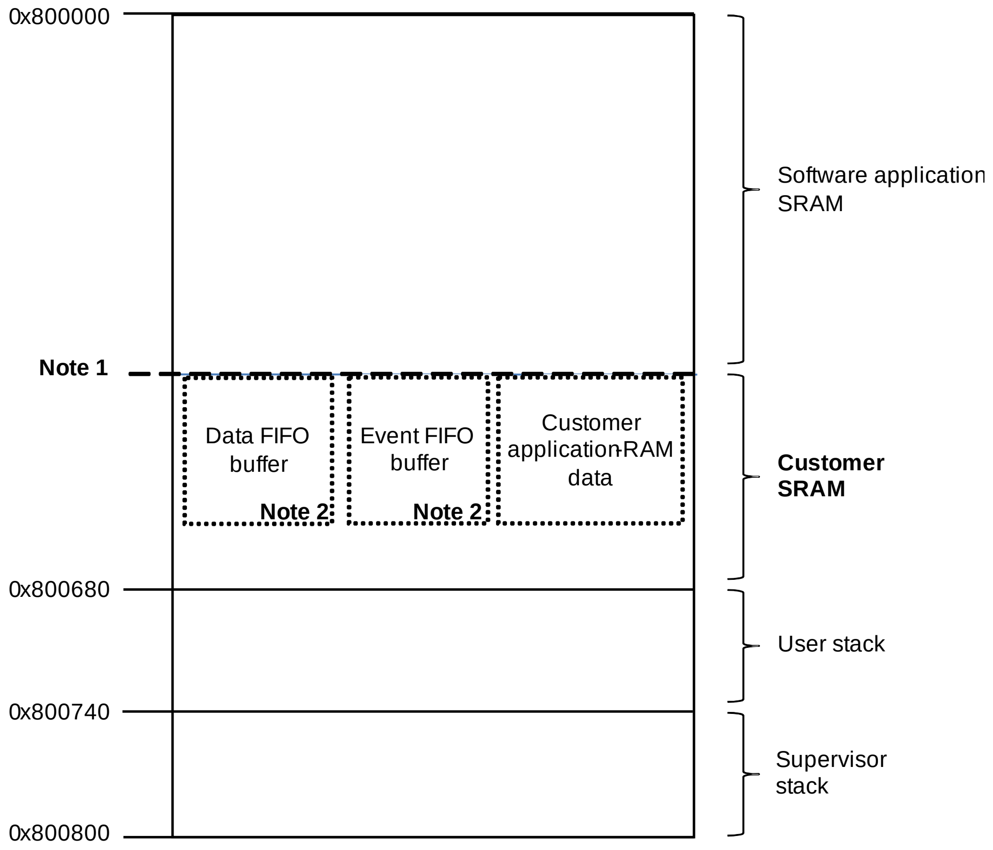
Chapter 20 Memory Allocation for User Applications

20.1 Overview of memory allocation

User applications can allocate their data variables such as status outputs, configuration inputs and private variables. The RAM memory distribution of the MMA955xL device is divided in four sections:

- Firmware SRAM
- User SRAM (Data FIFO, Event FIFO, and user application data)
- User stack
- Supervisor stack.

The memory distribution is shown in the following figure.



2. For the actual ending address of the software application SRAM, see the release notes.
3. These two buffers are allocated dynamically in the user's SRAM space. The start and end address of each buffer depends on which one is allocated first.

Figure 20-1. RAM memory map

The distribution of the user RAM section will start from the end address of the “software application SRAM” section. The actual starting address varies, depending on which device in the MMA955xL family is being used. The location of each user block (Data FIFO buffer, Event FIFO buffer, and User Variables) depends on the order that the memory blocks were requested.

The RAM memory is reserved in one direction which means that is not possible to free memory. The user should avoid using static memory defined by a linker file. If this approach is used, care must be taken to not override memory allocated through the API calls.

The definition of the user data structure must start with the outputs, followed immediately by the parameters—without any padding in between—and the private variables as is shown in the following example:

Example 20-1.

```
typedef struct user_filter_struct_tag {
struct user_filter_outs_tag {
int16 xyz[USER_FILTER_AXES];
uint8 counter;
} outs;
struct user_filter_param_tag {
uint8 max_boundary;
uint8 min_boundary;
} param;
struct user_filter_private_tag {
buffer_t buffer[USER_FILTER_AXES];
} private;
} user_filter_struct_t;
```

20.2 API functions

The Freescale-platform firmware provides two API functions for user applications to request and reserve RAM memory:

uint8_t * RequestDataRam (uint16_t sz, uint8_t u8appid)

Description: This function reserves the size of memory passed as a parameter starting from the last RAM address used. The application data-start address that is reserved is registered into the global application data that is used by the communication interface to configure the parameter and read-output structures.

Parameters:

- u8Size – Number of memory bytes to reserve by the function.
- uint8_t u8appid – The identifier of the application, with valid values [0 to 31]. The identifier value cannot duplicate values used by the firmware applications

Return: void * - Pointer to the start address of the reserved memory. If not enough memory is available, NULL is returned.

void * GetDataPtr (uint8_t u8appid)

Description: This function returns the pointer to the application data-structure address. The function must be called when a user application must access data variables that were not assigned at compilation time.

Parameters: uint8_t u8appid – The application's identifier.

Return: void * – Pointer to the application data structure. If memory was not previously reserved for the specific APP_ID, a NULL value is returned.

For details on the indices of the previous functions, in the Freescale functions table, see "User Applications" on page 197.

Chapter 21 User Applications

21.1 Application binding

The MMA955xL platform contains a user flash memory section reserved for up to three user images. User images contain an identification table and functions associated with a user application. User applications can be executed by the scheduler in the Freescale firmware.

Freescale firmware provides a mechanism for downloading up to three user images and binding them into the primary image. A user application must have the defined application table as a header, as shown in Figure 21-1 on page 198.

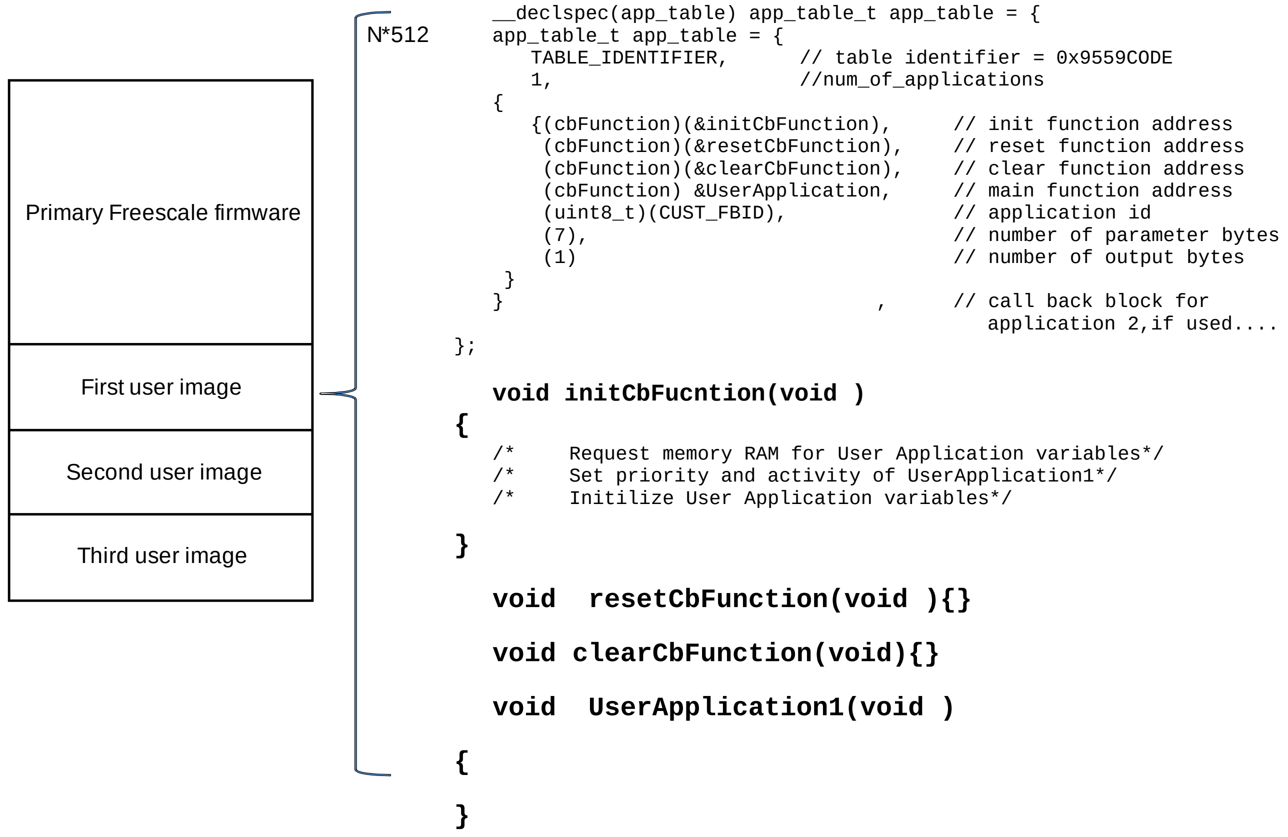
In addition to the main user function, applications must also have associated callback functions for init, reset, and clear. These three callback functions allow the scheduler, as well as the host and other applications, to control applications.

The init function should allocate and initialize all variables that are used by the application. The reset function should put the application in a state to be started or re-started. The clear function should clear out all the output or status results of the application.

Each image can have multiple applications.

Application tables must be located at specific locations in the flash memory so that the bootloader can find them. In the programming application note and associated templates, the compiler options and linker file are configured to properly locate the application tables at appropriate memory locations.

The following image shows the flash memory map with the Freescale and user memory sections:



```
      __declspec(app_table) app_table_t app_table = {
      app_table_t app_table = {
         TABLE_IDENTIFIER,        // table identifier = 0x9559CODE
         1,                       //num_of_applications
      {
         {(cbFunction)(&initCbFunction),      // init function address
          (cbFunction)(&resetCbFunction),     // reset function address
          (cbFunction)(&clearCbFunction),     // clear function address
          (cbFunction) &UserApplication,      // main function address
          (uint8_t)(CUST_FBID),               // application id
          (7),                                // number of parameter bytes
          (1)                                 // number of output bytes
       }
      }                                ,      // call back block for
                                                 application 2,if used....
   };

      void initCbFucntion(void )
   {
      /*     Request memory RAM for User Application variables*/
      /*     Set priority and activity of UserApplication1*/
      /*     Initilize User Application variables*/

   }

      void  resetCbFunction(void ){}

      void clearCbFunction(void){}

      void  UserApplication1(void )

   {

   }
```

Figure 21-1. Flash memory map

The binding mechanism, which happens at boot time, requires that a user-image application table be located at the beginning of a new page of 512 bytes. If the table is at another address, the binding process will not able to find the user images.

The application table starts with a 32-bit table identifier which is 0x9550C0DE. The application table also contains the number of applications and the parameters of each application, in the order to be executed by the firmware scheduler.

The initialization function in the user application needs to reserve and initialize the data memory for public and private variables. It also needs to configure the priority and activity of the application to be executed by the scheduler.

The reset function in the user application needs to do the things needed to reset the user application. This function is used by the system and possibly other applications to reset the user application.

The clear function in the user application clears the output bytes of the user application. This function is used by the system and possibly other applications to clear the outputs of the user application.

Figure 21-1 shows an example of the structure of a user images with just one application. The total number of applications is limited to 31, including the Freescale applications. The number of user images is limited to three. The Freescale firmware uses x of the tasks, leaving x applications for users to implement in their systems.

The application identifiers used by the Freescale firmware are documented in "Communication Interface" on page 55.

21.2 API

The MMA955xL device's firmware provides an API for the user application code. These API functions can be accessed through the instruction in Assembler Trap *7*. The index of the desired function is in Register D0.

The indexes of the Freescale API table are described in the following enumeration:

```
typedef enum fsl_api_indx_tag {
  FSL_API_FN_EVNT_Q             = 0,      /*eventFifo_push */
  FSL_API_FN_REG_USER_INT       = 1,      /*register_user_int */
  FSL_API_FN_REQ_DATA_RAM       = 2,      /*RequestDataRam */
  FSL_API_FN_GET_APMAP_PTR      = 3,      /*get_apmap_ptr */
  FSL_API_FN_APP_OUT_ADDR       = 4,      /*app_output_addr */
  FSL_API_FN_APP_PARAM_ADDR     = 5,      /*app_param_addr */
  FSL_API_FN_IIR_FILTER         = 6,      /*iir filter*/
  FSL_API_FN_GET_DATA_PTR       = 7,      /*GetDataPtr*/
  MAX_FSL_API
}fsl_api_indx_tag;
```

The definitions of the indexes, in the preceding enumeration, are:

FSL_API_FN_EVENT_Q –

Index of the function that pushes data into the Event FIFO.

FSL_API_FN_REG_USER_INT –

Index of the function that registers users' interruption functions.

FSL_API_FN_REQ_DATA_RAM –

Index of the function that dynamically reserves RAM.

FSL_API_FN_GET_APMAP_PTR –

Index of the function that gets the address of an entry in the application table.

FSL_API_FN_APP_OUT_ADDR –

Index of the function that looks for, and returns the address of, the status registers for a given APP_ID.

FSL_API_FN_APP_PARAM_ADDR –

Index of the function that looks for, and returns the address of, the configuration registers for a given APP_ID.

FSL_API_FN_IIR_FILTER –

Index of the configurable IIR filter function.

FSL_API_FN_GET_DATA_PTR –

Index of the function that looks for, and returns the address of, the application data.

21.3 Additional resources

For additional information on developing custom applications, see:

- *Installation of the MMA955xL CodeWarrior Service Pack* (AN4128)
- *Build Custom Applications on MMA9550/MMA9551L* (AN4129).

To access the webpage with these documents, see "References" on page 13.